НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Програмний засіб зберігання наукових праць за технологією блокчейн»

Виконав (-ла):
студент (-ка) IV курсу, групи ІТ-51
Давиденко Ігор Володимирович _____

Керівник:
ст. викладач Мітін Сергій В'ячеславович _____

Рецензент:
Доцент спеціальної кафедри №5 ІСЗЗІ КПІ ім. Сікорського,
к.т.н. Цуркан В.В.
_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 рік

ЗМІСТ

ВСТУП

Зберігання наукових праць є головною задачею на сьогодні, оскільки, завершивши написання наукової роботи, виникає потреба в опублікуванні документа так, щоб він не був у подальшому змінений або видалений іншими користувачами системи. Як результат, для забезпечення зберігання даних користувача, використовують різні методи для розробки таких систем і однією з них є технологія блокчейн.

Для реалізації застосунку, зберігання даних у мережі блокчейн, було обрано реалізацію системи у вигляді веб-затосунку, оскільки системи котрі розроблені з використанням веб є платформо не залежними, забезпечують своєчасне виправлення всіх помилок у створеній системі, в результаті чого користувачі, не залежачи від платформи, завжди матимуть доступ до системи.

Завданням дипломного проекту є розробка веб-застосунку, який буде надавати наступні можливості користувачу системи:

- перегляд сторінок;
- перегляд документів;
- коментування документів;
- додавання документів.

Метою проекту є розробка веб-застосунку, який матиме можливість зберігати наукові праці студентів, які у подальшому можна буде опублікувати в мережі блокчейн, що в результаті забезпечить користувачу цілісність його даних.

Предметом дослідження дипломного проекту є системи розподіленого зберігання даних та методи реалізації зберігання наукових робіт із використанням технології блокчейн.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Першим етапом розробки програмного забезпечення є аналіз та визначення його предметної області. Розгляд практичності і актуальності обраної теми є головним завданням при розробці програмного забезпечення.

Також для більшої ясності було розроблено декілька UML діаграм, які в певній мірі показують функціонал створеної системи. Таким чином ми можемо спроектувати поведінку та визначити, які саме функції повинна забезпечувати розроблювана система.

1.1 Аналіз інформаційних потреб та визначення предметної області

Першим етапом розробки системи є аналіз інформаційних потреб, тому на цьому етапі розробки програмного продукту визначається, що повинна забезпечувати розроблювана система.

Система повинна бути розроблена у вигляді веб-застосунку, оскільки саме такий вигляд системи дає можливість користуватися нею з різних платформ, якою кожного дня будуть послуговуватись велика кількість користувачів. Застосунок у свою чергу повинен бути розміщений на веб-сервері, де відбуваються всі запити до бази-даних і генеруються веб-сторінки з результатом запиту. Також система повинна надавати можливість користувачу системи взаємодіяти з платформою блокчейн. Таким чином, виконавши описані вище вимоги, систему можна буде вважати завершеною.

Цей програмний засіб розробляється як система, котра повинна зберігати документи користувачів на сервері з можливістю їх подальшого розміщення в системі блокчейн. Оскільки вона буде орієнтована на зберіганні даних, а саме документів, потрібно буде продумати алгоритми шифрування та стиснення даних. Також система повинна мати захист від несанкціонованого доступу зі сторони зовнішнього світу, а саме це взлом за допомогою ботів, підбору паролів, sql ін'єкцій

та інших методів злому веб-ресурсів. Також вона повинна гарантувати, що поштовий адрес користувачів буде захищений від попадання в руки спам ботів.

Наступною головною частиною будь-якої системи є її адміністрування, і тому реалізована система повинна мати можливість доступу до застосунку від імені її адміністратора, котрий у свою чергу може моніторити дії, котрі виконують користувачі системи, та в разі необхідності переглядати стан системи блокчейн, щоб у випадку падіння, перезапустити її.

Користувачам системи має бути надано можливість виконувати базові дії, такі як редагування, додавання, видалення. Першою і головною є операція редагування, яка включає в себе можливість редагувати дані користувацького акаунту, дані документів, котрі були додані користувачами системи, також коментарі, котрі користувачі залишили під тим чи іншим документом.

Друга базова операція - це додавання даних, котра включає в себе такі дії, як: користувач системи має можливість реєструватися на сайті, додавати нові документи, коментарі та робочі етапи з питаннями.

Третя базова операція - це видалення, яка включає в себе можливість видалити користувача, документ, коментарі, питання.

Процедура додавання документів до блокчейну матиме декілька етапів, кожний з яких передбачає такі дії, як: підготовка документу, його публікація на сайті, набір підписів користувачів системи та його публікація на вузол блокчейну. Першим етапом є підготовка документу та його публікація, для цього користувач повинен тільки створити документ та опублікувати його, наступним етапом є те, що наукова робота користувача збирає деяку кількість підписів (більше 10) користувачів даної системи, після чого буде сформована транзакція, яка додає блок із даними до блокчейну. Саме ці дії повинен виконати користувач системи, щоб додати до блокчейну свою наукову роботу.

Також одним із важливих процесів є процес написання документу, тому для того щоб користувач міг покращувати свої документи до відповідного рівня за допомогою інших користувачів системи, то їм буде надана можливість коментувати документи, створювати чат прив’язаний до документа, в якому можуть

обговорювати цю роботу. Чати у свою чергу також надають можливість створювати бесіди з іншими користувачами сайту.

Для користувачів буде введено класову систему, яка полягатиме в розділенні користувачів на такі типи, як: користувач, студент, аспірант, доцент, професор, доктор наук. У кожному із типів діятиме рейтингова система, яка буде розміщувати за рейтингом користувачів (який буде визначатися в залежності від кількості опублікованих робіт у мережі блокчейн) та їх роботи, надасть можливість відслідковувати місце знаходження їх у списках, крім цього користувач системи може переглядати список всіх робіт, які будуть ділитися на опубліковані та неопубліковані. Серед яких також буде діяти рейтингова система, яка буде показувати найбільш компетентні роботи у відповідній галузі. Наукові роботи також будуть розподілятися по галузям, що дасть можливість упорядкувати інформацію, оптимізувати та пришвидшити пошук.

Таким чином розроблювана система повинна задовольнити всі потреби користувачів при написанні їх документів та полегшити процес його розробки, та публікації в науковому світі.

1.2 Діаграма використання (Use case)

При розробці системи діаграма використання є однією з головних її частин, оскільки саме вона відповідає на одне з головних питань: які саме дії повинна виконувати система в реальному світі. При розробці цієї діаграми можна використовувати 2 типи основних сутностей: варіанти використання(додавання, редагування, видалення і т.д.) і дійові особи(користувач системи), між якими можна встановити зв'язки типу асоціація.

Зв'язок асоціація надає можливість встановити відповідність між дійовою особою та варіантом її дії в процесі використання системи. Тобто асоціація показує, що особа, котра буде користуватися розробленою системою, в будь-якому випадку бере участь у виконанні кожного із зазначених сценаріїв, які описані варіантом використання. Зв'язок асоціації вважається одним із важливих і практично завжди

обов’язковим зв'язком при проектуванні діаграми використання. У випадку відсутності цього типу зв'язку, можна вважати, що систему, над якою ведуться розробки, ніяким способом не взаємодіє з навколишнім світом.

Як видно із діаграми зображеної на додатку А, основними діями користувача є Перегляд, Додавання, Видалення, Редагування. Саме ці дії лежать в основі кожного розробленого застосунку, оскільки вони є базовими, їх користувач може виконати користуючись застосунком. Першою операцією, яку здійснює користувач системи, є додавання, яка в собі включає вибір предметної області документу та категорії, до якої відноситься документ, і останнім етапом, для того щоб додати новий документ, є вибір та завантаження потрібного документа. Після виконання описаних дій користувач може створити новий документ.

Після створення документа і публікації його на сайті, користувач має можливість переглядати створені документи. Під час цієї він має можливість залишити коментарі під документом.

Також для створеного документу є можливість додавати питання для документів.

1.3 Діаграма діяльності

Для опису послідовності дій користувача системи застосовано діаграму діяльності. Дана діаграма розроблена для опису послідовності дій системи на основі переданих повідомлень. У процесі розробки діаграми під дією розуміється особа, яка використовується для визначення поведінки або специфікації. Кожна із дій даної діаграми може отримувати множину вхідних сигналів та перетворювати їх на множину вихідних сигналів. Кожна із зазначених дій в діаграмі діяльності може виконуватись один, два, або більше разів під час одного виконання. Кожна зі зазначених дій має отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати від проектувальника певної послідовності. Тож для більшої ясності процесу додавання документа було спроектовано діаграму діяльності зображену на додатку Б.

Як видно з додатку Б, для створення нового документа користувачу системи потрібно першим етапом - це вибрати категорію документа, потім він повинен обрати предметну область документа та заповнити додаткові поля форми. Після виконання попередніх дій користувач системи повинен завантажити файл та натиснути кнопку підтвердити, після чого відбувається валідація файлу, у випадку відповідності валідуючим правилам, дані будуть збережені, в іншому - їх буде повернено на сторінку додавання документу, де він повинен повторити вище описані дії.

Отже, цей тип діаграми надає можливість зрозуміти послідовність дій користувача системи при користуванні нею. Як результат у розробника системи є можливість більш краще зрозуміти, які саме елементи інтерфейсу потрібно буде розробити.

1.4 Діаграма послідовності

У процесі розробки програмного продукту виникає потреба планування послідовність взаємодії об'єктів впорядкованих за часом, тому на поміч приходить діаграма послідовності. Цей тип діаграми показує потік повідомлень між об'єктами програмного продукту. Кожний із діаграм послідовностей складається із об'єктів, котрі позначаються прямокутниками з підкресленими іменами, асоціації між вказаними об'єктами позначаються у вигляді з'єднувальних ліній, над ними може бути зображена стрілка із зазначенням назви повідомлення та його порядкового номера.

Для більш кращого розуміння, як буде відбуватися обмін повідомленнями між компонентами розроблюваної системи, було спроектовано діаграму діяльності, котра знаходиться на додатку В.

Як видно з додатку В, ця система буде складатися з таких основних об'єктів, як: користувач, який являє собою діючу особу, що буде виконувати всі основні дії із системою, панель користувача, яка надає користувачу системи базові дії, які він може виконувати із системою, наступним об'єктом є сайт та сам блокчейн.

1.5 Діаграма компонентів

Під час розробки системи завжди виникає потреба в описі фізичного представлення системи. Даний тип діаграм надає можливість розробнику системи визначити її архітектуру, яка розробляється в цей момент часу, встановити залежності між компонентами програми, якими може бути як початковий так і виконуваний код. При проектуванні даного типу діаграми розробник користується елементами діаграми: компоненти та залежності між ними. Одними з основних цілей даного типу діаграми є можливість візуалізувати структуру початкового коду програмної системи, забезпечити подальше багатократне використання окремих частин програмного коду, також представити концептуальні і фізичні схеми баз даних. Діаграма компонентів розроблюваної системи зображена на рисунку 1.4.

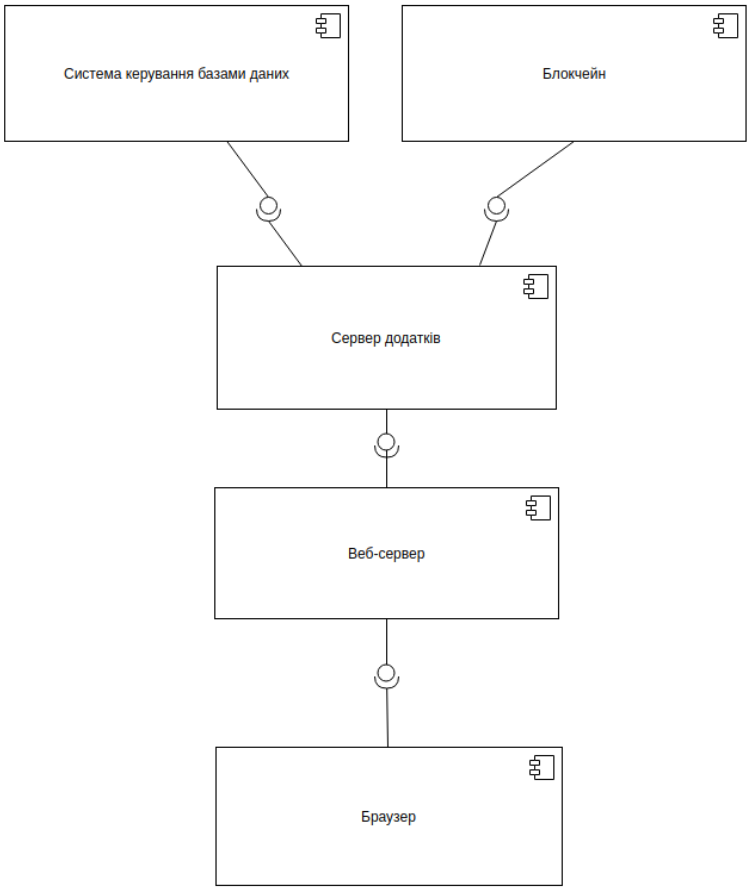

Рисунок 1.4 - Діаграма компонентів системи

Діаграма компонентів, зображена на рисунку 1.4, відображає нам послідовність з'єднання компонентів у цьому програмному продукті. Першим

компонентом є браузер, оскільки користувачу для взаємодії з системою потрібно відкрити веб-сторінку в браузері. У процесі завантаження відбувається взаємодія з наступним компонентом, котрий являє собою веб-сервер, роль якого в даній системі полягає в обробці даних та підключенні до двох компонентів, серед яких є система керування базами даних та блокчейном.

У результаті проектування діаграми компонентів системи, виділились такі основні частини, як: система, котра керує базами даних, блокчейном, сервер додатків, веб-сервер та браузер, за допомогою якого користувач буде взаємодіяти з системою.

1.6 Актуальність

Важливу роль при розробці наукової роботи відіграє етап її написання і публікації у світовій мережі або в друкованому вигляді. І на етапі публікації документу в електронному вигляді виникає потреба у виборі системи, яка буде забезпечувати збереженість даних. Одним із рішень є ela.kpi.ua, яка надає можливість зберігати, переглядати наукові роботи, котрі були написані студентами КПІ. На відміну від наведеної системи розроблений програмний продукт дає можливість користувачам системи не тільки опублікувати наукові роботи, а також коментувати та обговорювати їх.

Головною метою цього проекту являється розробка системи, котра буде задовольняти потреби її користувачів та надавати їм можливість зберігати данні у відповідній системі з гарантом того, що вони будуть у подальшому не змінними, та наступним етапом розробки продукту буде перехід на dapp архітектуру веб-застосунку.

Таким чином даний програмний продукт надає можливість студентам, аспірантам та другим науковим діячам опублікувати їхню працю, з можливістю її наступного перегляду, що в подальшому допоможе їм підтвердити свій науковий рівень.

1.7 Практична цінність

У процесі розробки системи однією з головних частин є її практична цінність, оскільки визначення області використання системи є одним із головних етапів розробки системи та подальшого її розвитку.

Головною метою проекту є забезпечення користувачів системи можливістю написати та опублікувати їх наукові роботи, що у свою чергу дозволяє користувачам системи покроково спроектувати етапи написання наукових робіт і також, аналізуючи коментарі, швидко виправляти помилки, зроблені при написанні документа.

Розроблена система створюється з орієнтацією на те, що нею будуть користуватися студенти, аспіранти, доктори наук і т.д., які пишуть наукову роботу і хочуть, щоб їхня праця не була забута та послужила основою для майбутніх проектів.

Таким чином головною практичною цінністю цієї системи є її можливість використовувати в усіх областях, де потрібно, документувати свої дослідження з їх подальшим опублікуванням у світову мережу.

Висновки до розділу

У даному розділі було проведено аналіз предметної області програмного забезпечення, також визначено актуальність та практичність розроблюваної системи, що при розробці систем даного типу є головним завданням при проектуванні.

Також для більшої ясності було спроектовано та розроблено декілька UML діаграм, які в певній мірі показують, яким чином буде працювати розроблюваний продукт.

2. ВИБІР ТА ОБҐРУНТУВАННЯ ЕЛЕМЕНТІВ ТА ТЕХНОЛОГІЙ

У процесі розробки системи головним етапом є вибір та обґрунтування обраних технологій, з якими в подальшому буде працювати розробник системи, тому основною із цілей даного розділу є пояснення того чи іншого вибору проектувальника системи. Оскільки саме вибір технологій лежить в основі подальшого розроблення програмного продукту.

2.1 Системи розподіленого зберігання даних

Розподілена система зберігання даних включає в собі можливості зберігання даних на різних вузлах комп'ютерної мережі, як результат дані зберігаються не на одному сервері, а на декількох. Цей підхід проектування має декілька характеристик, серед яких виділяються наступні:

- неоднорідність;
- розподіленість;
- автономність.

Кожна із наведених характеристик в певній мірі відображає, якою саме повинна бути розподілена система зберігання даних. Першою характеристикою є неоднорідність системи зберігання даних, котра включає в себе дві або більше істотно різних продуктів управління даними (наприклад, реляційні СУБД від різних постачальників, таких, як Oracle і Postgresql, або СУБД одного постачальника, але функціонують на різних платформах і використовують різні структури баз даних, такі, як DB2 і SQL / DS компанії IBM), таким чином розроблювана система не буде залежати тільки від одного продукту управління даними, що в подальшому допоможе застосовувати різні підходи до рішення поставлених задач при розробці застосунку.

Наступною характеристикою цих баз даних є розподіленість, а саме є розподіл компонентів даних для зберігання між комп'ютерами мережі.

І останньою характеристикою системи розподіленого зберігання даних є автономність, котра полягає в можливості системи зберігання даних автономно виконувати свої функції і в разі виникнення проблем на одному із вузлів мережі, замінити її.

На сьогоднішній день одним із популярних варіантів використання розподіленого зберігання даних є блокчейн, котрий на відміну від розподілених баз даних завжди відсутній центральний адміністратор, який конфігурує вузли мережі, тому виходить, що архітектура блокчейну не просто розподілена, але й децентралізована. У ньому для підтримки актуальності даних на її вузлах діють так звані алгоритми для забезпечення консенсуса в даній мережі.

Завдання розподіленого консенсусу не специфічна тільки для блокчейна і має добре перевірені рішення для багатьох інших розподілених систем (наприклад, баз даних NoSQL).

Але блокчейн біткойну й інші реалізації блокчейну від попередніх напрацювань відрізняються умовами роботи мережі. У звичайних алгоритмах візантійського консенсусу у вузлів мережі є «особистості», що виражаються через цифрові підписи або подібні криптопримітиви, а сам список вузлів відомий заздалегідь або змінюється рідко, але передбачувано. У випадку блокчейну біткоїну все працює навпаки.

Учасники мережі не тільки заздалегідь невідомі, але і можуть вільно підключатися або відключатися від мережі. При цьому блокчейн, будучи децентралізованою системою, має певні властивості: стійкість до цензури (ніхто не може заборонити майнити криптовалюту) і об'єктивність (для визначення поточної версії журналу транзакцій не потрібно довіряти якимось авторитетним джерелам - корінь довіри знаходиться в самому блокчейні).

Через це звичайні алгоритми візантійського консенсусу для блокчейну не підходять. Тому було запропоновано безліч різних алгоритмів, серед яких виділяються дві основні категорії: алгоритми на основі доказу роботи (proof-of-work) і алгоритми на основі підтвердження частки (proof-of-stake).

2.1.1 Алгоритми візантійського консенсусу

Даний тип консенсусу є одним із задач криптографії, завдання якого полягає в рішенні проблеми взаємодії кількох віддалених абонентів, які отримали накази з єдиного центру.

Формулювання задачі полягає в наступному: кожен із генералів має отримати один і той же вектор довжини n, у котрому і-й елемент або містить чисельність і-ї армії (якщо її командувач лояльний), або невизначений (якщо командувач — зрадник). У результаті дослідження даної проблеми виникло дві математичні моделі — інтерактивна система доказів і доказ із нульовим розголошенням.

Інтерактивну систему доказів (P, V, S) слід розуміти як протокол взаємодії двох абонентів: P (той що доводить) та V (той що перевіряє). Абонент P хоче довести V, що твердження S істинне. Абонент V самостійно, без допомоги P, не може довести твердження S (тому V й називають перевіряльником). Абонент P може бути й ворогом, котрий хоче довести V, що твердження S істинне, хоча воно хибне[18].

У визначенні системи (P, V, S) не припускається, що V може бути ворогом. Протокол (P, V, S), що розв'язує таку задачу, у свою чергу називається доказом із нульовим розголошенням і повинен відповідати, окрім умов 1 і 2, ще такій умові, що нульове розголошення (або стійкість) — у результаті роботи протоколу (P, V, S) абонент V не збільшує свої знання про твердження S або, іншими словами, не зможе здобути ніякої інформації про те, чи S істинне[18].

2.1.2 PoW(proof-of-work) консенсус

Proof-of-work являється одним із найпопулярніших алгоритмів консенсусу на сьогодні, оскільки даний алгоритм здобуття згоди між вузлами мережі використовується в блокчейні криптовалюти біткоїн. Даний алгоритм підтримання консенсусу отримав цілком академічну назву - консенсус Накамото. Автор даного алгоритму Сатоши Накамото запропонував «підписувати» кожен створюваний блок

доказом роботи, складність якого залежить від загальної обчислювальної складності мережі біткойн.

Докази роботи використовуються вузлами біткойнів для визначення стану системи. Актуальний журнал транзакцій визначається як ланцюжок блоків із найбільшою сумарною складністю доказів роботи. Майнери, відповідно, повинні шукати блок поверх цього ланцюжка[19]. Але, теоретично, ніхто не забороняє створювати нові блоки на основі якогось старого блоку (іноді трапляються розщеплення - форк - блокчейну, тому що два майнера знаходять новий блок практично одночасно). Однак навмисний форк невигідний економічно, тому що блоки з побічних гілок блокчейна ніким не враховуються і не приносять їхнім творцям ніякого прибутку, а одні витрати на перебування доказу роботи. А за рахунок того, що докази роботи швидко перевіряються і не вимагають для перевірки нічого, крім блокчейну, досягається об'єктивність протоколу.

2.2 Методи стиснення та кодування даних

Кодування та стиснення інформації на сьогоднішній день є головною частиною при розробці системи, котра буде зберігати великі об’єми даних, оскільки попри збільшення об'єму накопичувачів, зберігання інформації все ще залишається актуальною проблема більш ефективного використання пам'яті. Більш корисним є послуговування алгоритмів стиснення при передача інформації в системах зв'язку. У останньому випадку з'являється можливість передавати значно меншу (як правило, у декілька разів) кількість даних, що за собою несе менші ресурси пропускної спроможності каналів для передачі тієї ж самої інформації.

Оскільки головною задачею є зберігання документів у блокчейні, то перед тим як записувати документ до блоку, його потрібно зжати. Для цього обраний один із алгоритмів стиснення даних та кодування інформації. Для кодування файлу обраний один із найпростіших алгоритмів кодування інформації Base 64. Алгоритм кодування за допомогою Base 64 використовується для представлення довільних послідовностей символів у формі, яка дозволяє використовувати як верхній, так і

нижній регістр, у результаті чого не повинні бути читабельними для людини, що дозволить захистити передаваємі дані.

Процес кодування інформації, котру хоче закодувати користувач системи, представляє собою набір 24-бітових груп вхідних бітів, у результаті перетворення яких на виході одержуються рядки з 4 кодованими символами. Виконуючи зліва направо, 24-бітна група вхідних символів формується шляхом об'єднання з 8-бітних груп вводу[11].

Ці 24 біти потім обробляються як чотири об'єднані 6-бітові групи, кожна з яких переводиться в єдиний символ у базовому алфавіті 64. Кожна 6-бітна група використовується, як індекс у масиві з 64 друкованих символів. Символ, на який посилається індекс, поміщається у вихідний рядок[11].

Спеціальна обробка виконується, якщо в кінці кодуються даних менше ніж 24 біта. Повний кінець кодування завжди завершується наприкінці величини. У випадку коли в групі вводу доступно менше 24 вхідних бітів, додаються біти з нульовим значенням (праворуч), щоб сформувати ціле число 6-бітових груп. Заповнення вкінці даних виконується за допомогою символу '='. Таким чином, використовуючи даний тип кодування, ми можемо будь-яку інформацію представити у вигляді набору символів будь-якої послідовності.

2.3 Обґрунтування вибору засобів реалізації.

У процесі розробки програмного забезпечення рано чи пізно виникає проблема зберігання даних, і для вирішення цієї проблеми, і були розробленні так звані бази даних.

Бази даних являють собою спеціально розроблене сховище для зберігання різних типів даних. Кожна база даних, має певну модель (реляційна, документно-орієнтована), яка забезпечує зручний доступ до даних[3].

Системи управління базами даних (СУБД) - спеціальні додатки (або бібліотеки) для управління базами даних різних розмірів і форм[4].

Також за моделлю даних виділяють такі види СУБД :

– ієрархічні;
– мережеві;
– реляційні;
– об'єктно-орієнтовані;
– об'єктно-реляційні.

Нижче перераховані основні функції СУБД:

– визначення даних - визначити, яка саме інформація зберігатиметься в базі даних, задати властивості даних, їх тип (наприклад, число цифр або символів), а також вказати, як ці дані зв'язані між собою, також у деяких випадках є можливість задавати формати і критерії перевірки даних;
– обробка даних - дані можуть оброблятися самими різними способами, і можна вибирати будь-які поля, фільтрувати і сортувати дані, надається можливість об'єднувати дані з іншою пов'язаною з ними інформацією і обчислювати підсумкові значення;
– управління даними - можна вказати, кому дозволено ознайомитися з даними, коректувати їх або додавати нову інформацію, також для адміністратора системи надається можливість також визначати правила колективного доступу.

Аналіз та вибір СУБД проведено з урахуванням того, що система керування базами даних повинна витримувати високу нагруженість, буди простою в використанні, мати можливість взаємодіяти з нею використовуючи технологію клієнт-сервер, також однією з важливих вимого до обраної бази даних є підтримка нею стандарту SQL.

При виборі потрібно також враховувати, що число клієнтських місць становить до 600000, а доступ до даних має бути максимально ефективним. Обчислювальна техніка працюватиме під керуванням операційної системи, котра використовує ядро Linux.

У процесі порівняння при створенні проекту використовувалися три наступні бази даних : MySQL, Oracle та PostgreSQL. Кожна з вибраних баз даних є реляційною, що відповідає базовим вимогам, тому далі проводиться порівняння

обраних баз даних, що в подальшому надасть можливість більш детальніше розглянути всі недоліки та переваги той чи іншої обраної бази даних.

Кожна із обраних баз даних має в собі функції, що і притаманні іншим базам даних, серед яких можна виділити наступні:

- локалізація інтерфейсу користувача, можливість побудови і сортування полів БД, що містять символи кирилиці;
- підтримка структури відносних даних;
- підтримка технології клієнт/сервер;
- підтримка багатопроцесорної архітектури;
- підтримка кластерної архітектури;
- наявність засобів для створення індексів і кластерів для підвищення ефективності використання даних;
- відновлення баз даних із використанням журналу транзакцій;
- механізм блокування транзакцій під час запису або на рівні сторінок;
- підтримка ANSI SQL;
- підтримка стандарту SQL-3 (нова назва – SQL99);
- контроль цілісності БД;
- підтримка ODBC;
- підтримка утиліт резервування БД;
- імпорт/експорт таблиць БД;
- сумісність з ОС модулів користувача та сервера;
- підтримка визначених мережевих протоколів;
- наявність графічного інтерфейсу для адміністраторів БД;
- підтримка служби єдиного каталогу.

На основі порівняння обраних баз даних проведено експертну оцінку багатокористувацьких СУБД, що в подальшому надасть можливість більш краще зрозуміти яку саме базу даних вибирають експерти, для реалізації більш громіздких проектів.

Результати проведення експертного оцінювання наведенні в таблиці 2.1.

Таблиця 2.1 – Експертна оцінка багатокористувацьких СУБД

СУБД	Продуктив-ність	Конкурент-ний доступ	Кількість користува-чів	Великі БД	Готовність
MySQL	8	8	8	7	8
Oracle	6	8	8	7	7
PostgreSQL	7	8	8	8	8

Наступним етапом порівняння реляційних баз даних є виділення основних переваги та недоліки кожної з наведених вище СУБД. У результаті якого в подальшому з’являється можливість розглянути, які є переваги та недоліки в обраних базах даних, що в результаті допоможе з її вибором.

Переваги PostgreSQL:

- відкрите ПЗ відповідає стандарту SQL – база даних є безкоштовною з відкритим вихідним кодом. Дана СУБД є дуже потужною системою;
- велике співтовариство - спільнота даної бази даних досить велика, у якій запросто можна знайти відповіді на всі свої питання;
- велика кількість доповнень – окрім величезної кількості вбудованих функцій, існує дуже багато доповнень, що дозволяють розробляти дані для цієї СУБД і управляти ними;
- розширення – надає можливість розширювати функціонал, за рахунок збереження своїх створених процедур;
- об'єктність – окрім того що дана СУБД є реляційною, але також вона об'єктно-орієнтована з підтримкою успадкування і багато іншого.

Недоліки PostgreSQL:

- продуктивність - при простих операціях читання PostgreSQL може значно уповільнити роботу сервера і поступитися цим своїм конкурентам, таким як MySQL;

– популярність - за своєю природою, популярністю ця СУБД похвалитися не може, хоча і є досить велика спільнота;
– хостинг - у силу названих вище чинників іноді досить складно знайти хостинг із підтримкою цієї СУБД.

Переваги Oracle:

– oracle - це потужний сервер, призначений для корпоративних цілей.

Недоліки Oracle:

– платна система.

Переваги MySQL:

– простота в використанні – процес встановлення MySQL проходить досить просто, також розроблені різноманітні програми, наприклад Workbench, який дозволяє досить легко підключатися та працювати з базою даних;
– багатий функціонал - MySQL підтримує більшість конструкцій мови декларативного програмування SQL;
– безпека – має в наявності велику кількість функцій, що забезпечують безпеку, серед яких привілеї для користувачів, запобігання ін'єкцій SQL та пошкодження даних;
– масштабованість – база даних легко працює з великими обсягами даних і легко масштабується, також підтримує кластеризацію;
– швидкість - спрощення деяких стандартів дозволяє MySQL значно збільшити продуктивність.

Недоліки MySQL:

– відомі обмеження - за задумом у MySQL закладені деякі обмеження функціонала, які іноді необхідні в особливо вимогливих додатках;
– проблеми з надійністю - через деяких способів обробки даних MySQL (зв'язку, транзакції, аудити) іноді поступається іншим СУБД по надійності;
– повільна розробка - хоча MySQL технічно відкрите ПЗ, існують скарги на процес розробки, також варто зауважити, що існують інші досить успішні СУБД, створені на базі MySQL, наприклад MariaDB[20].

Проаналізувавши вище описані недоліки та переваги з кожних систем керування базами даних, можемо підсумувати наступне, що з розглянутих типів СУБД найбільше нам підходить PostgreSQL, тому що він є безкоштовним, має більшу швидкість обробки запитів користувачів, стабільний і має хорошу систему користувачів, є доволі простим у налаштуванні. Також беручи до уваги обрану нами систему для розробки веб-застосунку – Spring Framework, вибір PostgreSQL являється логічним.

2.4 Обгрунтування вибору блокчейн платформи

У процесі проектування проекту на основі блокчейну одним з головних етапів є вибір платформи на основі якої буде відбуватися наступна розробка застосунку, тому наступним етапом є дослідження платформи для побудови веб-застосунків на основі технології блокчейн. Мета даного пошуку є вивір такої платформи, що надасть можливість писати контракти з використанням тільки однієї мови програмування, як результат в подальшому допоможе прискорити процес написання програмного продукту.

Відібрані наступні платформи для побудови блокчейну:

- Corda;
- Hyperledger Fabric;
- Ethereum;
- MultiChain.

Кожна з наведених технологій окрім MultiChain має в своїй наявності можливість реалізувати бізнес логіку у вигляді Chain Code або Smart Contract, який у свою чергу є фрагментом коду що написано на одній із підтримуваних мов, таких як Go або Java, Solidity, Rust, C#, Python т.д. Chain Code сам пособі встановлюється і реалізується через SDK або CLI в мережу рівноправних вузлів вибраної платформи, яка дозволяє взаємодіяти з загальною книгою цієї мережі.

В процесі роботи з блокчейном дуже часто фігурує визначення вузла яке зазвичай може значно змінюватися в залежності від контексту. Коли мова йде про

комп'ютерні або телекомунікаційні мережі, вузли можуть пропонувати різні цілі, діючи або як точка перерозподілу, або як кінцева точка зв'язку. Як правило, вузол складається з фізичного мережевого пристрою, але є певні випадки, коли використовуються віртуальні вузли. Простіше кажучи, мережевий вузол - це точка, де може бути створено, отримано або передано повідомлення.

Наступним дуже важливим поняття при роботі зі блокчейном є книжка мережі. Книжка мережі (або розподілена книга) — це база даних, яка поширюється по декільком вузлам або обчислювальних пристроїв[1]. Також кожен вузол мережі відповідає таким умовам як вузол реплікує зберігає ідентичну копію книги та вузол мережі оновлюється самостійно.

Таким чином блокчейн є однією з форм розподіленої книги. Де всі розподілені регістри використовують ланцюжок блоків для забезпечення безпечного і надійного розподіленого консенсусу.

Блок-ланцюг розподіляється між мережами тимчасових мереж і управляється ними. Оскільки дана мережа є розподіленим регістром, то він може існувати без централізованого управління або сервера, керуючого їм, а його якість даних може підтримуватися реплікацією бази даних і обчислювальною довірою.

Однак структура блокового ланцюга робить її відмінною від інших розподілених регістрів. Дані про блокової ланцюжки групуються в блоки. Потім блоки з'єднуються один з одним і захищаються з використанням криптографії.

Блок-ланцюжок по суті є постійно зростаючим списком записів. Його структура тільки для додавання і дозволяє додавати дані в базу даних: змінити або видалити раніше введені дані на більш ранніх блоках неможливо. Тому технологія блокчейн підходить для запису подій, ведення записів, обробки транзакцій, відстеження активів і голосування. Як результат дана технологія і набула своєї популярності в роботі з валютами і зберіганні даних.

Першою платформою є Ethereum, який відомий по однойменній криптовалюті котра є другою за популярністю після Bitcoin. Ethereum - платформа яка на даний час використовується для створення децентралізованих онлайн-сервісів на базі блокчейна, бізнес логіка яких реалізована у вигляді розумних контрактів. Вузли в

даній платформі реалізовані у вигляді єдиної децентралізованої віртуальної машини. Мовою для написання смарт контрактів в даній мережі використовують Solidity, котрі транслюються в байткод EVM. Однією з особливостей даної мови програмування є те що мова «Тьюрінг-повна», тобто підтримує більш широкий набір обчислювальних інструкцій. З чого слідує, що набір правил для маніпуляції з даними, котрі прописані в ядрі мови програмування відповідають всім вимогам щоб вважати її вважається повним за Тюрінгом.

В результаті аналізу даної платформи виявлено недолік, який полягає в оплаті транзакцій, що в результаті робить неможливим створити програмний продукт який буде безкоштовним як для користувачу так і розробнику, тому вона не відповідає вимогам для вибору платформи на якій будуть зберігатися користувацькі документи.

Наступним на черзі у нас буде MultiChain, цікава платформа для розробки застосунків на основі блокчейну з хорошим набором бібліотек на різних мовах програмування. Однією з особливостей даної платформи є те що вона надає можливість будувати приватний блокчейн та взаємодіяти з ним за допомогою RPC, що в свою чергу відкриває можливість не обмежуватись мовою програмування, а писати власні бібліотеки для взаємодії з нею. На відміну від попередньої платформи в даній відсутні смарт-контракти в будь якому вигляді, як результат платформа не задовольняє поставлені вимоги.

Corda представляє собою вид платформи на базі Ethereum з обмеженим доступом, смарт-контракти якої побудовані на JVM, використання платформи можливо виключно фінансовими установами. Corda являє собою децентралізовану базу даних або розподілений реєстр. При цьому створені проекти говорять і про інших можливих застосуваннях даної платформи.

Головною особливістю платформи є те, що вона не використовує блок. Замість цього застосовуються спеціальні нотаріальні ноди. Сама по собі, Corda не використовує концепцію майстерності і систему Proof-of-Work. У найближчому розглянутій платформі найбільш схоже з концепцією баз даних Bigchain.

Однією особливістю даної платформи є можливість розробляти так званні CorDapps (Corda Distributed Applications). Завдання яких полягає в тому, щоб дозволити вузлам досягти домовленості про оновлення головної книги. Вони досягають цієї мети, визначаючи потоки, які власники вузлів Corda можуть викликати через RPC.

CorDapps мають форму набору файлів JAR, що містять визначення класів, написані на Java або Kotlin.

Зазвичай визначені класи складаються з наступних елементів:

- потоки: визначають процедуру для запуску вузла, як правило, для оновлення головної книги. Для реалізації потоку потрібно унаслідувати клас FlowLogic;
- стани: визначають факти, щодо яких досягнуто згоди. Для реалізації станів потрібно реалізувати інтерфейс Contract State;
- контракти, що визначають, що є дійсним оновленням книги. Для реалізації контракту потрібно реалізувати інтерфейс Contract;
- послуги, що забезпечують довговічні сервіси в межах вузла. Для реалізації послуг потрібно унаслідувати клас Singleton Serialization Token;
- білі списки серіалізації, обмежуючи типи, які вузол отримуватиме по дроту. . Для реалізації білих списків потрібно реалізувати інтерфейс Serialization Whitelist;

Corda цікава платформа для розробки блокчейну, але на момент розробки дана платформа сильно орієнтувалася на Kotlin, і таким чином не відповідає вимогам на відміну наступного учасника відбору.

Наступною платформою є одна з популярних платфом яка в свою чергу має хороший набір фреймворків для розробки додатків на основі блокчейну - Hyperledger.

Hyperledger це один з проектів The Linux Foundation який несе в собі ідею що тільки спільний підхід до розробки програмного забезпечення з відкритим вихідним кодом може забезпечити прозорість, довговічність, сумісність та підтримку, необхідні для приведення технологій blockchain для впровадження комерційного

впровадження. Саме про це йдеться в Hyperledger - спільнотах розробників програмного забезпечення, які будують фреймворки і платформи для блокчейн.

Для реалізації проекту застосовується Hyperledger Fabric. Hyperledger Fabric - це реалізація блокчейна і один з проектів Hyperledger, розміщених у Linux Foundation. Основним з призначень є основою для розробки додатків або рішень з модульною архітектурою, Hyperledger Fabric дозволяє компонентам, таким як консенсус і послуги членства, бути plug-and-play[1]. Hyperledger Fabric використовує технологію контейнера для розміщення інтелектуальних контрактів, які називаються "ланцюговим кодом" (chaincode), що складається з логіки застосування системи. Hyperledger Fabric спочатку розробляли Digital Asset і IBM, в результаті першого хакатона.

Для написання смарт-контрактів використовують: Go, Java, Js. Для розробки користувальницького додатка, Hyperledger Fabric надає SDK, яке дозволяє взаємодіяти з її нодами.

На вузлах виконується бізнес логіка (смарт-контракт) - chaincode, зберігається стан розподіленого реєстру (ledger data) і виконуються інші системні служби платформи.

Додаток користувача (Submitting Client) - додаток, за допомогою якого користувачі працюють з блокчейн мережею. В свою чергу повинен пройти авторизацію і володіти відповідними правами на різного роду дії в мережі. Для авторизації використовується виділений центр сертифікації - CA (Certification Authority). CA працює на основі X.509 стандарту та інфраструктури публічних ключів — PKI. Також одними з головних елементів є канал — це закрита підмережа, що складається з двох або більше учасників блокчейн мережі, призначена для проведення конфіденційних транзакцій всередині обмеженого, але відомого, кола учасників. Канал визначається учасниками, своїм розподіленим реєстром, смарт-контрактами, Ordering Service, WorldState. Кожен учасник каналу повинен бути авторизований на доступ до каналу і мати право виконувати різного роду транзакції. Авторизація виконується за допомогою Membership Service.

Таким чином виділяються наступні елементи:

- для написання смарт контракту ми можемо використовувати Golang, JAVA, Node.js;
- для взаємодії з нодою предоставляють SDK, для таких мов програмування як node.js, java, go;
- для виконання дій в блокчейні кожен учасник каналу повинен бути авторизований.
- можливість створення закритої підмережі котра називається каналом.

В результаті аналізу вище описаних бібліотек для побудови блокчейну, обрано Hyperledger Fabric. Оскільки даний фреймворк надає можливість розробляти клієнтський додаток та chaincode з використанням мови програмування Java, також надає можливість захисту даних від несанкціонованого доступу, котрі будуть зберігатися в блокчейні.

Висновки до розділу

В даному розділі було обрано технології з використанням яких буде розроблятись подальша система та детальне описання кожної із обраних технологій з якими в подальшому буде працювати розробник системи. Оскільки саме вибір технологій лежить в основі подальшого розроблення програмного продукту.

3. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА БІЗНЕС-ЛОГІКИ ПРОЕКТУ

При розробці системи одним із головних етапів є проектування архітектури системи, оскільки на її основі буде виконуватися вся подальша розробка програмного забезпечення. Якісна розробка системи надає розробнику можливість швидко виконувати розробку програмного продукту і також допомагає усунути помилки викликані не правильним проектуванням системи в процесі її розробки.

Першим етапом при розробці системи є аналіз інформаційних процесів котрі відбуватимуться в розроблюваній системі.

3.1 Аналіз інформаційних процесів

Аналіз інформаційних процесів є однією із найважливіших частин при розробці застосунку, оскільки завдяки аналізу процесів серед яких одержання, зберігання й передачі інформації можливо зрозуміти як саме буде працювати розроблювана система. Таким чином завдяки аналізу інформаційних процесів можливо мати уявлення про інформацію котра отримується в результаті дій, що були виконані над системою.

При проектуванні системи першим етапом постає задача розробити структурну схему системи, яка допоможе чітко виділити модулі з якими в подальшому відбуватиметься робота. Використовуючи структурну схему можна розділити систему на сукупність елементарних ланок об'єктів і зв'язками між ними, та також дана діаграма в свою чергу являється одним із видів графічної моделі. Під елементарною ланкою використовується поняття частини системи управління задача якої полягає в реалізації елементарної функції. Таким чином використовуючи структурну схему розробник системи матиме можливість зрозуміти які потрібно буде розробити модулі в процесі написання застосунку.

Структурна схема розроблюваної системи для зберігання файлів зображена на рисунку 3.1.

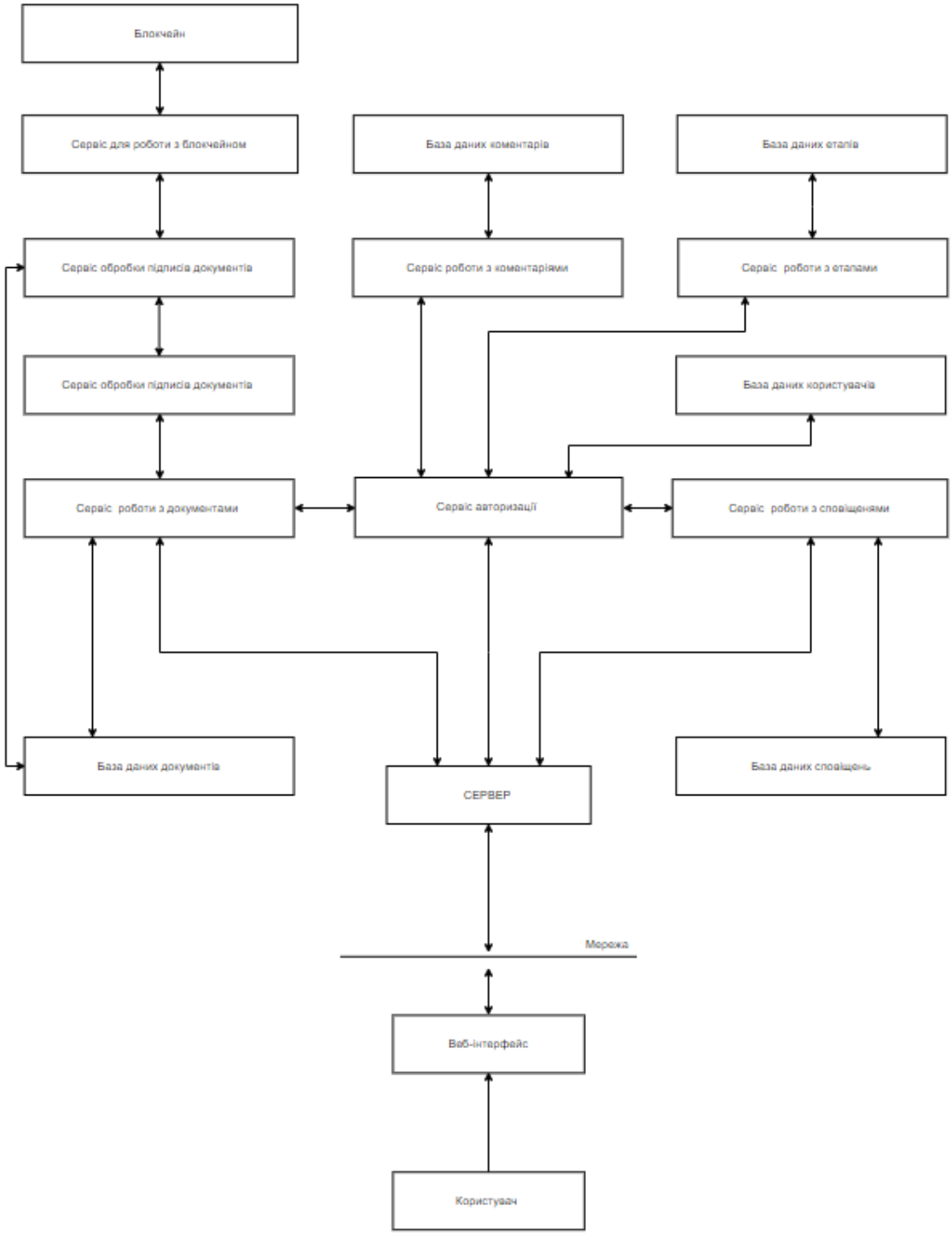

Рисунок 3.1 – Структура інформаційної системи

Як видно з рисунку 3.1 майбутня система буде складатися з серверу та клієнтською частиною. Сервер в свою чергу буде містити вісім сервісів кожний із яких виконує певні поставлені задачі.

Одним із головних елементів розроблюваної системи є сервер, окільки завдяки йому відбувається обробка даних користувача системи, котрі він передає за

допомогою інтерфейсу користувача, котрий в свою чергу є головним компонентом через який відбувається взаємодія із системою, завдяки неї користувач може здійснювати, всі дії які надав йому розробник системи.

Другим і най значимими елементами системи є сервіси системи, за допомогою яких і здійснюється взаємодія з базою в результаті чого сервером відбувається генерація представлення котре виводиться в подальшому користувачу системи в інтерфейсі застосунку. Одним із головних сервісів системи є сервіс авторизації, який надає користувачам можливість авторизуватися в системі та виконувати потрібні дії, також даний сервіс відіграє одну із най важливіших ролей в розроблюваній системі, оскільки для виконання певних дій кожний з сервісів повинен спочатку зробити запит на отримання авторизованого користувача і тільки потім виконувати поставлені задачі.

Третьою частиною системи є сервіс для взаємодії з блокчейном, даний сервіс представляє розробнику набір функцій для підключення до мережі блокчейну та здійснення потрібних дій. Даний сервіс є частиною системи сервісів за допомогою яких відбувається робота зі документами користувачів системи, одним із таких сервісів є сервіс який обробляє підписи документів при роботі якого відбувається взаємодія з базою даних документів для отримання їх списку.

Для зображення послідовності виклику послідовності виклику компонентів, системи створена функціональна схема застосунку зображена на рисунку 3.2.

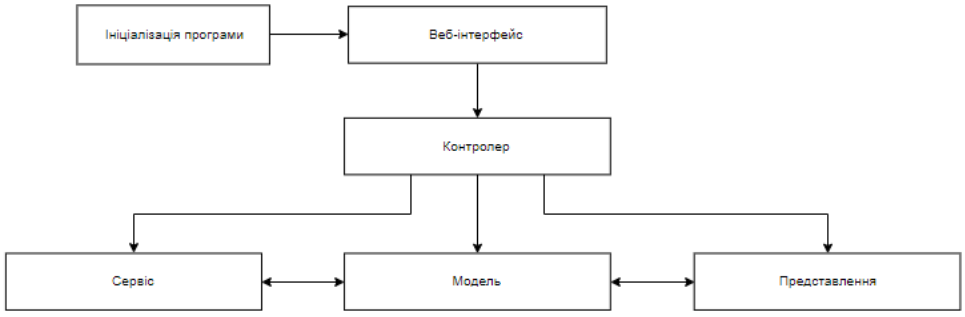

Рисунок 3.2 - Функціональна схема застосунку

На рисунку 3.2 зображена функціональна схема застосунку. Котра в свою чергу відображає процедури, які виконуються при виникненні певних подій

розроблюваної системи. Після ініціалізації застосунку користувач системи потрапляє в веб-інтерфейс котрий надає можливість здійснювати користувачу потрібні задачі в процесі виконання яких відбувається генерації запиту на сервер, на сторінці серверу для обробки запиту користувача відбувається виклик відповідного сервісу котрий взаємодіє з моделями і в результаті генерується представлення які виводяться користувачу в веб-інтерфейсі застосунку.

3.2 Розробка моделі інформаційної системи

В процесі проектування програмного продукту, модель інформаційної системи являється одним із головних елементів розроблюваної системи, оскільки завдяки моделі проектувальник системи має можливість відобразити послідовність взаємодії компонентів системи. Для зображення послідовності виклику компонентів системи розроблено схему, яка зображена на рисунку 3.3.

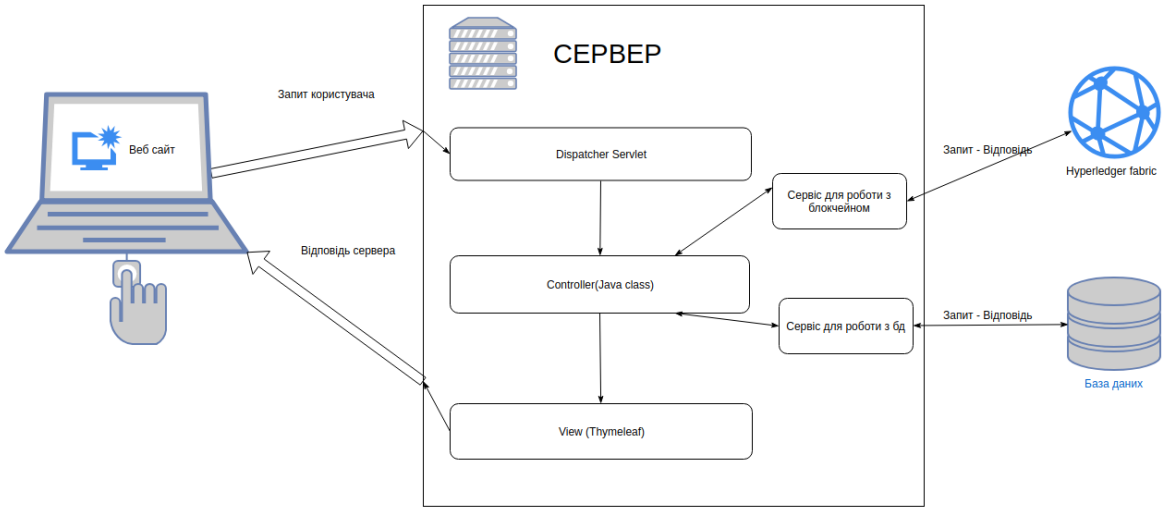

Рисунок 3.3 - Деталізований алгоритм роботи програмного продукту

Як видно з рисунку 3.3 розроблена система матиме трьох вимірну архітектуру котра є стандартну для більшості не великих систем в сфері WEB-розробки. Дана архітектура складається з клієнту, серверу та бази даних. Клієнт формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними,

що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних. У порівнянні з клієнт-серверною або файл-серверною архітектурою можна виділити наступні переваги трирівневої архітектури:

- масштабованість;
- конфігурованість — ізольованість рівнів один від одного дозволяє (при правильному розгортанні архітектури) швидко і простими засобами пере конфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів[5];
- високий рівень безпеки;
- висока надійність;
- низькі вимоги до швидкості каналу (мережі) між терміналами і сервером застосунків;
- низькі вимоги до продуктивності і технічних характеристик терміналів, як наслідок зниження їхньої вартості, і терміналом може виступати не тільки комп'ютер, але і, наприклад, мобільний телефон[4].

При розробці веб додатків даного типу архітектури одним з головних патернів розробки на даний момент часу залишається Model-View-Controller. MVC - це тип проектування системи в основі якого лежить принцип розділення проекту на три головні частини серед яких ми можемо виділити є Контролер, Модель та Представлення[7].

Також даний патерн проектування веб систем передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: Модель, Представлення і Контролер – в результаті чого модифікація кожного компонента може здійснюватися незалежно, що зробить розробку програмного забезпечення більш продуктивнішим.

Також основною метою застосування цієї концепції полягає в відділенні бізнес-логіки (моделі) від її візуалізації (уявлення, виду). За рахунок такого поділу

підвищується можливість повторного використання коду. Найбільш корисне застосування даної концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах і з різних точок зору. Зокрема, виконуються наступні завдання:

- до однієї моделі можна приєднати кілька видів, при цьому не зачіпаючи реалізацію моделі;
- не торкаючись реалізацію видів, можна змінити реакції на дії користувача (натискання мишею на кнопці, введення даних) - для цього досить використовувати інший контролер;
- ряд розробників спеціалізується тільки в одній з областей: або розробляють графічний інтерфейс, або розробляють бізнес-логіку, тому можливо добитися того, що програмісти, які займаються розробкою бізнес-логіки (моделі), взагалі не будуть обізнані про те, яке уявлення буде використовуватися.

3.3 Проектування бази даних системи

При розробці веб-застосунку однією з головних частин є проектування бази даних. База даних являється головною частиною проекту оскільки всі основні функції, а саме зберігання даних виконує саме вона. Саме для виконання всіх цих завдань мною було обрано PostgreSql котра надає користувачу дуже великі можливості для зберігання їх даних.

Першою частиною нашої розробки є проектування бази тож для початку потрібно виділити основні сутності які будуть фігурувати в базі даних.

При розробці бази даних веб-застосунку було створено наступні моделі:

- scr_users;
- scr_document;
- scr_author;
- scr_block_documents;
- scr_comment_issue;

- scr_comment_likes;
- scr_document_comments;
- scr_document_likes;
- scr_document_types;
- scr_documents_types;
- scr_document_subject;
- scr_document_subjects;
- scr_documents_work_flow;
- scr_user_document_signification;
- scr_roles;
- scr_isue_comments.

Для того щоб збереження даних про користувачів системи була розроблена таблиця «scr_users». Дана таблиця складається дванадцяти полів, перше з яких є первинним чисельним ключем, який слугує для ідентифікації користувача в системі. Інші поля з яких складається таблиця призначені для зберігання інформації про користувача. Склад створеної таблиці наведено в таблиці 3.1.

Таблиця 3.1 - Структура таблиці «scr_users»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор користувача
Uid	NVARCHAR	-	-	Адрес користувача згенерирований хеш функцією
Username	NVARCHAR	-	-	Логін користувача
Email	NVARCHAR	-	-	Адреса електронної пошти
Role	NVARCHAR	-	-	Роль користувача в системі
Password	NVARCHAR	-	-	Пароль
FirstName	NVARCHAR	-	-	Ім’я користувача
LastName	NVARCHAR	-	-	Прізвище користувача
isConfirmed	BOOLEAN	-	-	Перевірка чи є акаунт активованим

Продовження таблиці 3.1

banned	BOOLEAN	-	-	Перевірка чи є користувач заблокований
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_documents» розроблена для зберігання інформації про документи котрі були додані користувачами системи. Вона складається із десяти полів, перше з яких є первинним чисельним ключем, котре в свою чергу слугує для ідентифікації документу. Інші поля призначені для зберігання інформації документу. Склад створеної таблиці наведено в таблиці 3.2.

Таблиця 3.2 - Структура таблиці «scr_documents»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор користувача
Uid	NVARCHAR	-	-	Адрес користувача згенерирований хеш функцією
name	NVARCHAR	-	-	Назва документу
description	NVARCHAR	-	-	Короткий опис вмісту документу
path	NVARCHAR	-	-	Путь к файлу збереженого на сервері
watched	INT	-	-	Кількість переглядів файлу
stage	ENUM	-	-	Відображає етап публікації файлу
user_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_users полем id
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_author» розроблена для зберігання інформації про авторів системи, оскільки в розробці одного документу може приймати не тільки одна людина а декілька. Вона складається із семи полів, перше з яких є первинним

чисельним ключем, який також слугує для ідентифікації автора. Інші поля призначені для зберігання інформації про авторів документу. Склад створеної таблиці наведено в таблиці 3.3.

Таблиця 3.3 - Структура таблиці «scr_author»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор користувача
user_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_users полем id
document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id
role	NVARCHAR	-	-	Роль автора при написані документу
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_block_documents» розроблена для зберігання інформації про документи котрі були добавлені до блокчейну. Вона складається із п'яти полів, перше з яких є первинним чисельним ключем, який також слугує для ідентифікації документу. Інші поля призначені для зберігання інформації блоку. Склад створеної таблиці наведено в таблиці 3.4.

Таблиця 3.4 - Структура таблиці «scr_block_documents»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор користувача
hash	NVARCHAR	-	+	Нех адрес блоку в блокчейні
document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_comment_issue» розроблена для зберігання інформації про питання котрі виникли при розгляді документу. Вона складається із восьми полів, перше з яких є первинним чисельним ключем, який також слугує для ідентифікації питання. Інші поля призначені для зберігання інформації про питання. Склад створеної таблиці наведено в таблиці 3.5.

Таблиця 3.5 - Структура таблиці «scr_comment_issue»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор користувача
issue	NVARCHAR	-	-	Текст питання
commentType	ENUM	-	-	Тип питання
issueType	ENUM	-	-	Тип питання
comment_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_document_comments полем id
flow_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_document_work_flow полем id
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_comment_likes» розроблена для зберігання інформації про лайки, котрі були добавлені користувачами системи до коментарів. Вона складається із шести полів, перше з яких є первинним чисельним ключем, який також слугує для ідентифікації лайку доданого користувачем до коментарю. Інші поля призначені для зберігання інформації про лайки, котрі були додані користувачами системи до відповідного коментарю. Склад створеної таблиці наведено в таблиці 3.6.

Таблиця 3.6 - Структура таблиці «scr_comment_issue»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор користувача
likeType	ENUM	-	-	Тип лайку
comment_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_document_comments полем id
user_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_users полем id
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_document_comments» розроблена для зберігання інформації про коментарі додані користувачами системи до документу. Вона складається із дев'яти полів, перше з яких є первинним чисельним ключем, який також слугує для ідентифікації коментарю. Інші поля призначені для зберігання інформації коментарю. Склад створеної таблиці наведено в таблиці 3.7

Таблиця 3.7 - Структура таблиці «scr_document_comments»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор користувача
message	NVARCHAR	-	-	Текст коментарю
commentType	ENUM	-	-	Тип коментарю
commentState	ENUM	-	-	Стан коментарю
previous_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_document_comments полем id
user_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_users полем id

Продовження таблиці 3.7

document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_document_likes» розроблена для зберігання інформації про лайки до лайки документу. Вона складається із шести полів, перше з яких є первинним чисельним ключем, який також слугує для ідентифікації лайку доданого користувачем до документу. Інші поля призначені для зберігання інформації про лайки користувачів під документами. Склад створеної таблиці наведено в таблиці 3.8.

Таблиця 3.8 - Структура таблиці «scr_document_likes»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
likeType	ENUM	-	-	Тип лайку
user_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_users полем id
document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_document_types» розроблена для зберігання інформації про типи документів, що в подальшому допоможе здійснювати пошук потрібного документу набагато швидше. Вона складається із двох полів, перше з який є первинним чисельним ключем, який також слугує для ідентифікації типу. Інші поля призначені для зберігання інформації про типи документів. Склад створеної таблиці наведено в таблиці 3.9.

Таблиця 3.9 - Структура таблиці «scr_document_types»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
name	VARCHAR	-	-	Назва типу документа

Таблиця «src_documents_types» розроблена для створення зв’язку між документами та типами документу. Вона складається із трьох полів, перше з яких є первинним чисельним ключем, який також слугує для ідентифікації типу. Інші поля призначені для створення зв’язку між типами документів та самим документом. Склад створеної таблиці наведено в таблиці 3.10.

Таблиця 3.10 - Структура таблиці «src_documents_types»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
type_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_document_types полем id
document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id

Таблиця «scr_document_subject» розроблена для зберігання інформації про предметну область документу. Вона складається із двох полів, перше з яких є первинним чисельним ключем, який в свою чергу також слугує для ідентифікації предметної області. Інші поля призначені для зберігання інформації про предметну область. Склад створеної таблиці наведено в таблиці 3.11.

Таблиця 3.11 - Структура таблиці «scr_document_subject»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
Name	VARCHAR	-	-	Назва предмету

Таблиця «scr_document_subjects» розроблена для для створення зв’язку між документами та предметною областю документу. Вона складається із трьох полів, перше з яких є первинним чисельним ключем, який слугує для ідентифікації документу. Інші поля призначені для зберігання інформації посту. Склад створеної таблиці наведено в таблиці 3.12.

Таблиця 3.12 - Структура таблиці «scr_document_subjects»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
documentSubject_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_document_subject полем id
document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id

Таблиця «scr_documents_work_flow» розроблена для зберігання інформації про етапи написання документу. Вона складається із п'яти полів, перше з який є первинним чисельним ключем, який слугує для ідентифікації етапу. Інші поля призначені для зберігання інформації про етапи. Склад створеної таблиці наведено в таблиці 3.13.

Таблиця 3.13 - Структура таблиці «scr_documents_work_flow»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
Title	VARCHAR	-	-	Заголовок етапу
Document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id
ExpiresAt	DATE	-	-	Дата закінчення етапу
Stage	VARCHAR	-	-	Стан етапу

Продовження таблиці 3.13

Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_user_document_signification» розроблена для зберігання інформації про підписи користувачів під документами. Вона складається із семи полів, перше з яких є первинним чисельним ключем, який також слугує для ідентифікації підпису. Інші поля призначені для зберігання інформації підпис. Склад створеної таблиці наведено в таблиці 3.14.

Таблиця 3.14 - Структура таблиці «scr_user_document_signification»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
Note	VARCHAR	-	-	Зовнішній ключ який зв’язаний з таблицею scr_document_subject полем id
SignId	VARCHAR	-	-	Унікальний хеш для ідентифікації підпису згенерований хеш функцією
User_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_users полем id
Document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Таблиця «scr_roles» розроблена для зберігання інформації про ролі користувачів в системі. Вона складається із двох полів, перше з яких є первинним чисельним ключем, який слугує для ідентифікації ролі користувача в системі. Інші поля призначені для зберігання інформації. Склад створеної таблиці наведено в таблиці 3.15.

Таблиця 3.15 - Структура таблиці «scr_roles»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
Name	VARCHAR	-	-	Назва предмету

Таблиця «scr_issue_comments» розроблена для зберігання інформації про питання до документу. Вона складається із восьми полів, перше з яких є первинним чисельним ключем, який слугує для ідентифікації питання. Інші поля призначені для зберігання інформації про питання задане до коментарю. Склад створеної таблиці наведено в таблиці 3.16.

Таблиця 3.16 - Структура таблиці «scr_issue_comments»

Назва	Тип даних	ПК	ЗК	Опис поля
Id	INT IDENTITY	+	-	Ідентифікатор
Issue	VARCHAR	-	-	Загооловок питання
IssueType	VARCHAR	-	-	Тип питання
Comment_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_comments полем id
Document_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents полем id
Flow_id	INT	-	+	Зовнішній ключ який зв’язаний з таблицею scr_documents_work_flow полем id
Created_at	TIMESTAMP	-	-	Дата створення
Updated_at	TIMESTAMP	-	-	Дата змінення

Після того як було згенеровано список потрібних таблиць бази даних, наступним етапом проектування є проектування діаграми “сутність-зв'язок” котра дозволяє описувати концептуальні схеми за допомогою використання узагальнених

конструкцій та блоків. Існує ряд моделей для представлення даних, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення що його реалізує, є модель «сутність-зв'язок». Діаграма «сутність-зв'язок» зображена на додатку Г. На якій в свою чергу відображено зв'язок між розробленими моделями.

3.4 Проектування інтерфейсу обробки даних

Інтерфейс користувача є однією із головних частин при проектуванні системи тому було розроблено інтуїтивний інтерфейс, котрий складається з всіх необхідних форм для зручного користування системою. Таким чином наш інтерфейс буде складатися з таких трьох основних частин:

- звіти;
- форми;
- запити.

Також інтерфейс користувача буде містити всі необхідні налаштування для того щоб користувач системи зміг повноцінно скористатися розроблюваною системою цього були розробленні такі сторінки:

- головна сторінка;
- сторінка перегляду документу, та додавання нового коментарю;
- сторінка створення документу;
- сторінка додавання етапу написання та перегляду доданих;
- сторінка перегляду опублікованих документів;
- сторінка додавання питання та перегляду доданих;
- налаштування профіля користувача.

Для того щоб користувач системи мав можливість отримати всю необхідну інформацію було розроблено та протестовано такі наступні запити:

- змінити дані користувача;
- змінити дані документу;
- змінити текст коментаря;

– отримати перелік документів в блокчейні;
– отримати перелік коментарів створених користувачем;
– отримати перелік питань документів;
– отримати перелік коментарів в документі;
– отримати перелік документів створених користувачем.

Таким чином описані вище функції, повинні бути в подальшому реалізовані розробником системи. Реалізація вище описаного функціоналу в певній мірі забезпечить користувачу можливість повноцінно користуватись даною системою.

3.5 Реалізація операцій обробки даних в БД

Для роботи з базою даних використано Hibernate котра являється бібліотекою, яка розроблена для мови програмування Java, призначення якої полягає в звільненні розробника від великого обсягу низькорівневого програмування, в процесі роботи з об'єктно-орієнтованими засобами в реляційній базі даних. Розробник системи може використовувати Hibernate як в процесі проектування системи класів і таблиць «з нуля», так і для роботи з уже існуючою базою даних[2].

Бібліотека не тільки вирішує завдання зв'язку класів Java з таблицями бази даних (і типів даних Java з типами даних SQL), але і також надає засоби для автоматичної генерації і оновлення набору таблиць, побудови запитів і обробки отриманих даних і може значно зменшити час розробки, яке зазвичай витрачається на ручне написання SQL- і JDBC-коду[10]. Hibernate виконує автоматизацію генерації SQL-запитів і звільняє розробника від ручної обробки результуючого набору даних і перетворення об'єктів, максимально полегшуючи перенесення додатки на будь-які бази даних SQL.

Переваги які дає використання Hibernate:

– забезпечує простий API для запису та отримання Java-об'єктів в / з БД;
– мінімізує доступ до БД, використовуючи стратегії fetching;
– не вимагає сервера додатки;

– дозволяє нам не працювати з типами даних мови SQL, а мати справу з звичний нам типами даних Java;
– піклується про створення зв'язків між Java-класами і таблицями БД за допомогою XML-файлів не вносячи зміни в програмний код;
– якщо необхідні змінити БД, то достатньо лише ввести зміни в XML-файли.

Для роботи в Hibernate реалізована підтримка HQL (Hibernate Query Language) об'єктно-орієнтованої мови запитів, котра вкрай схожа з SQL. І однією з головних відмінностей HQL і SQL є те що SQL працює таблицями в базі даних і її стовпчику, а HQL - з збереженими об'єктами (Persistent Objects) і їх полями (атрибутами класу). Hibernate транслірує HQL запити в зрозумілі для БД SQL запити, які і виконують необхідні нам дії в БД. Також ми маємо можливість використовувати звичайні SQL - запити в Hibernate використовуючи Native SQL, але використання HQL є кращим[10]. Приклад використання HQL в нашій програмі зображений на наступному лістингу.

```
1  package com.igordavidenko.HouseNow.repo;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4  import org.springframework.data.jpa.repository.Modifying;
5  import org.springframework.data.jpa.repository.Query;
6  import org.springframework.data.repository.query.Param;
7  import org.springframework.stereotype.Repository;
8
9  import com.igordavidenko.HouseNow.Models.Users;
10
11 @Repository("userRepository")
12 public interface UsersRepository extends JpaRepository<Users, Long> {
13                         public Users findUserByUsername(String username);
14                         public Users findUserByEmail(String email);
15
16                                         @Modifying
17                                         @Query("UPDATE Users u SET u.username = :username,  u.email = :email, u.firstName = :firstName,
   u.lastName = :lastName, u.password = :password WHERE u.id = :id")
18                                         public int updateUser(@Param("id") Long integer, @Param("username")String username,
   @Param("email") String email, @Param("firstName") String firstName, @Param("lastName") String lastName, @Param("password") String
   password);
19
20                         @Modifying
21                         @Query("UPDATE Users u SET u.username = :username,  u.email = :email, u.firstName = :firstName, u.lastName =
   :lastName WHERE u.id = :id")
22                         public int updateUser(@Param("id") Long integer, @Param("username")String username, @Param("email") String email,
   @Param("firstName") String firstName, @Param("lastName") String lastName);
23 }
24
25
```

Лістинг 3.1 – Код класу «UsersRepository»

Даний HQL запит виконує оновлення даних користувача. Таким чином в прикладі використовуються анотація @Modyfying котра вказує користувачу що даний метод слідує розглядати як змінний запит, також використовується анотація

@Query котра дає можливість вказувати запити вручну. В наведеному прикладі використовується двокрапка перед назвою змінної, що в свою чергу дає можливість використовуючи анотацію @Param, котра змінює значення вказаної змінної. Таким чином відбувається обробка даних в даному дипломному проекті.

Також в даному дипломному проекті для взаємодії з даними в базі даних було реалізовано JPA (Java Persistence API) це специфікація Java EE і Java SE, що описує систему управління збереженням java об'єктів в таблиці реляційних баз даних в зручному вигляді. Сама Java не містить реалізації JPA, проте існує багато реалізацій даної специфікації від різних компаній (відкритих і немає). Це не єдиний спосіб збереження java об'єктів в бази даних (ORM систем), але один з найпопулярніших в Java світі.

Одним з прикладів реалізації є лістингу 3.1 де за допомогою Spring Data JPA котрий в свою чергу забезпечує модель програмування репозиторію, яка починається з інтерфейсу для об'єкта керованого домену, створено інтерфейс «UsersRepository» котрий є відкритим інтерфейсом що розширює JpaRepository <Users, Long> {...}. Визначення цього інтерфейсу служить двом цілям: по-перше, розширюючи JpaRepository, розробник отримує купу загальних методів CRUD для створеного типу, що дозволяє зберігати користувачів, видаляти їх і так далі. По-друге, це дозволить інфраструктурі сховища Spring Data JPA сканувати клас-шлях для цього інтерфейсу та створить для нього Spring bean.

Для того, щоб створити bean Spring, який реалізує цей інтерфейс, все, що потрібно зробити - це використовувати простір імен Spring JPA і активувати підтримку сховища за допомогою відповідного елемента.

Для реалізації самого веб-застосунку використовується Spring Boot, який надає можливість швидко розробляти веб-застосунки, що в результаті просто запускаються.

Серед переваг Spring Boot виділяються наступні:

- простота в розумінні коду та розробці застосунків;
- хороша продуктивність розроблених застосунків;
- зменшення часу розробки застосунку.

До недоліків даного фреймворку можна віднести:

- розроблені застосунки можуть включати в себе залежності, які не використовуються, в результаті чого одержуємо величезний розмір файлу розгортання;
- перенос Spring Framework застосунків на Spring boot вимагає багато зусиль та трудомісткого процесу;
- обмежений контроль розробленої програми.

Таким чином розробка програмного продукту з використанням Spring Boot надає можливість розробнику системи швидко розробляти та вносити нові функції в створений веб-застосунок.

3.6 Організація звітності системи

При проектуванні систем інформаційного типу, одним із головних елементів є звітність котра в певній мірі являє собою систему узагальнених показників, які характеризують діяльність програмної системи за певний період та генерація звіту зі значення показників сенсорів встановлених в відповідно до запиту її користувача. В даному випадку звітність складається на підставі даних зібраних із кількості переглядів сторінок користувачами системи, переглядів документів, та інформація про кількість лайків та підписів в документі і спираючись на отримані дані формуються звіти в формі веб-сторінок і в подальшому виводяться користувачу на екран в його персональній сторінці.

Користувач системи буде взаємодіяти з системою використовуючи так званні веб-сторінки, які формуються з боку серверу при переході на відповідне посилання в браузері користувача. Інтерфейс сайту в свою чергу є інтуїтивно зрозумілий та простий у використанні. При першому запуску програмного продукту користувач відразу потрапляє на головну сторінку, котра містить в правому верхньому куті навігаційне меню з використанням якого користувач має можливість переміщатися по сайту. Для того що авторизуватися в системі, користувач повинен перейти на сторінку зображеної на рисунку 3.4.

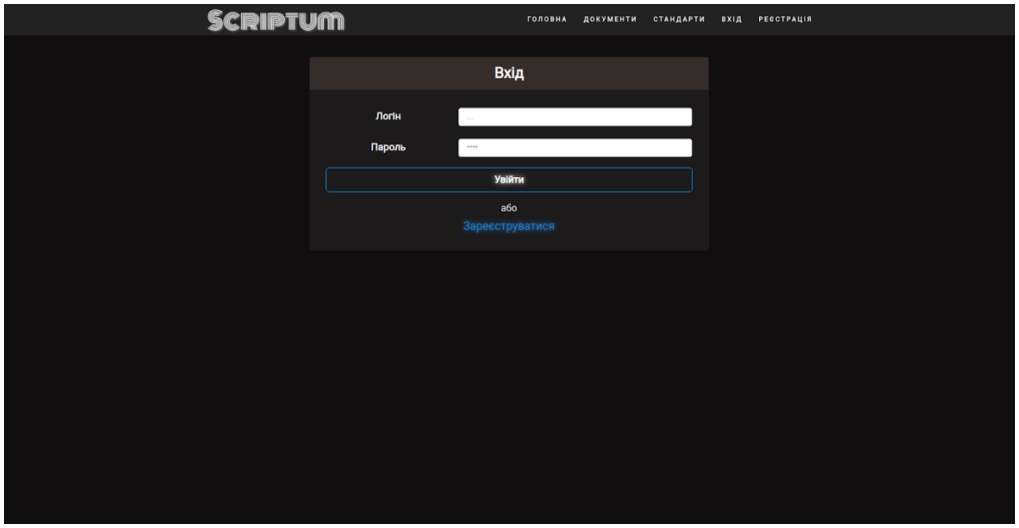

Рисунок 3.4 – Сторінка авторизації користувача

В випадку коли користувач системи є не зареєстрований і хоче потрапити на сторінку адміністратора чи зареєстрованого користувача його автоматично буде пере-направлено на головну сторінку сайту, з якої він може перейти на сторінку реєстрації чи увійти в створений профіль використовуючи форму авторизації яка зображена на рисунку 3.4.

В іншому випадку коли користувач авторизувався в системі, але не має відповідних привілеїв в системі, то його також буле пере направлено на головну сторінку застосунку. Після того як користувач увійде в свій профіль він матиме можливість створювати документи і залишати коментарі. У випадку коли користувач не зареєстрований у системі, він може зробити це, використовуючи форму, зображену на рисунку 3.5.

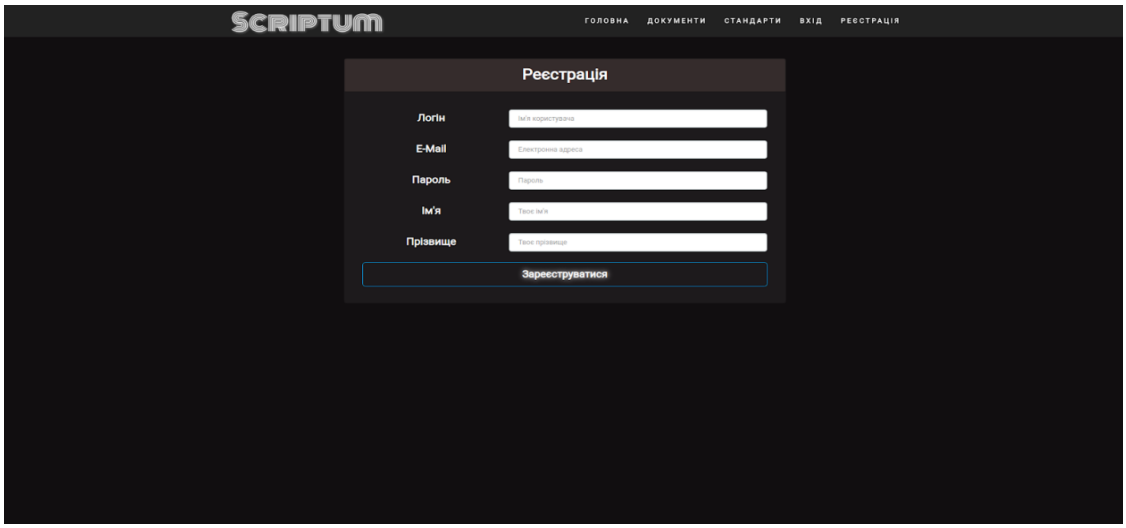

Рисунок 3.5 – Сторінка реєстрації користувача

Як видно з рисунку 3.5, користувачу системи, для того щоб зареєструватися на сайті потрібно буде вказати логін, почтову адресу, пароль, своє ім’я та прізвище. Після заповнення полів форми реєстрації та натисненні кнопки зареєструватися йому буде виведено повідомлення про успішну реєстрацію в системі.

Завершивши процес реєстрації, користувачу потрібно буде авторизуватися у системі, використовуючи вказаний при реєстрації логін та пароль, за допомогою форми зображеної на рисунку 3.4.

Після того як користувач авторизується на сайті він зможе переглядати документи, додавати коментарі або опублікувати свою власну наукову роботу.

Отже, аналізуючи вище описане в даному розділі можна сказати що звіти в розроблюваній системі будуть надавати користувачу завжди актуальну і детальну інформацію про документи котрі містяться на сайті.

Висновки до розділу

В даному розділі було проведено проектування архітектури подальшої системи, яке включає в собі побудову схеми “сутність-зв'язок”, проектування інтерфейсу обробки даних користувача та організовано звітність в системі.

4. РОЗРОБКА АРХІТЕКТУРИ МЕРЕЖІ БЛОКЧЕЙН

У процесі розробки веб-застосунку, що взаємодіє з мережею блокчейн виникає поняття архітектури мережі блокчейн. Під яким розуміється, що розробник системи повинен створити схематичний рисунок, котрий відображає яким чином будуть розміщені елементи даної мережі і зв'язок між ними, що надасть можливість у подальшому правильно налаштувати елементи даної мережі.

Оскільки, завдяки правильно спроектованій архітектурі мережі блокчейн надалі забезпечить цілісність збережених даних на вузлі мережі. Також правильне налаштування сприятиме швидшому досягненню консенсусу між вузлами мережі, що в подальшому надасть приріст у швидкості генерації блоків.

4.1 Проектування мережі з використанням Hyperledger Fabric

Поняття мережі при розробці блокчейн додатку є одним із головних, оскільки всі основні функції по збереженню даних лежать на вузлах цієї мережі, і тому розробка її архітектури є основною. Для розробки блокчейну було обрано фреймворк Hyperledger Fabric, який надає можливість створити власну приватну мережу.

Одним із головних елементів даної мережі є Peer. Peer - отримує впорядковані оновлення стану у вигляді блоків від сервісу замовлення і підтримує стан та книгу транзакцій[17]. Реалізація узгодження даних на різних вузлах полягає в тому, що спеціальна функція однорангового узгодження відбувається по відношенню до певного ланцюгового коду і затвердженні транзакції до її здійснення. Кожний ланцюговий код може вказувати політику схвалення, яка у свою чергу посилається на набір однорангових вузлів. Політика визначає необхідні та достатні умови для дійсного схвалення транзакції. У особливому випадку розгортання транзакцій, які встановлюють новий ланцюговий код, політика схвалення (розгортання) вказана як політика схвалення ланцюгового коду системи.

Приклад мережі з використанням двох peer сервісів зображена на рисунку 4.1.

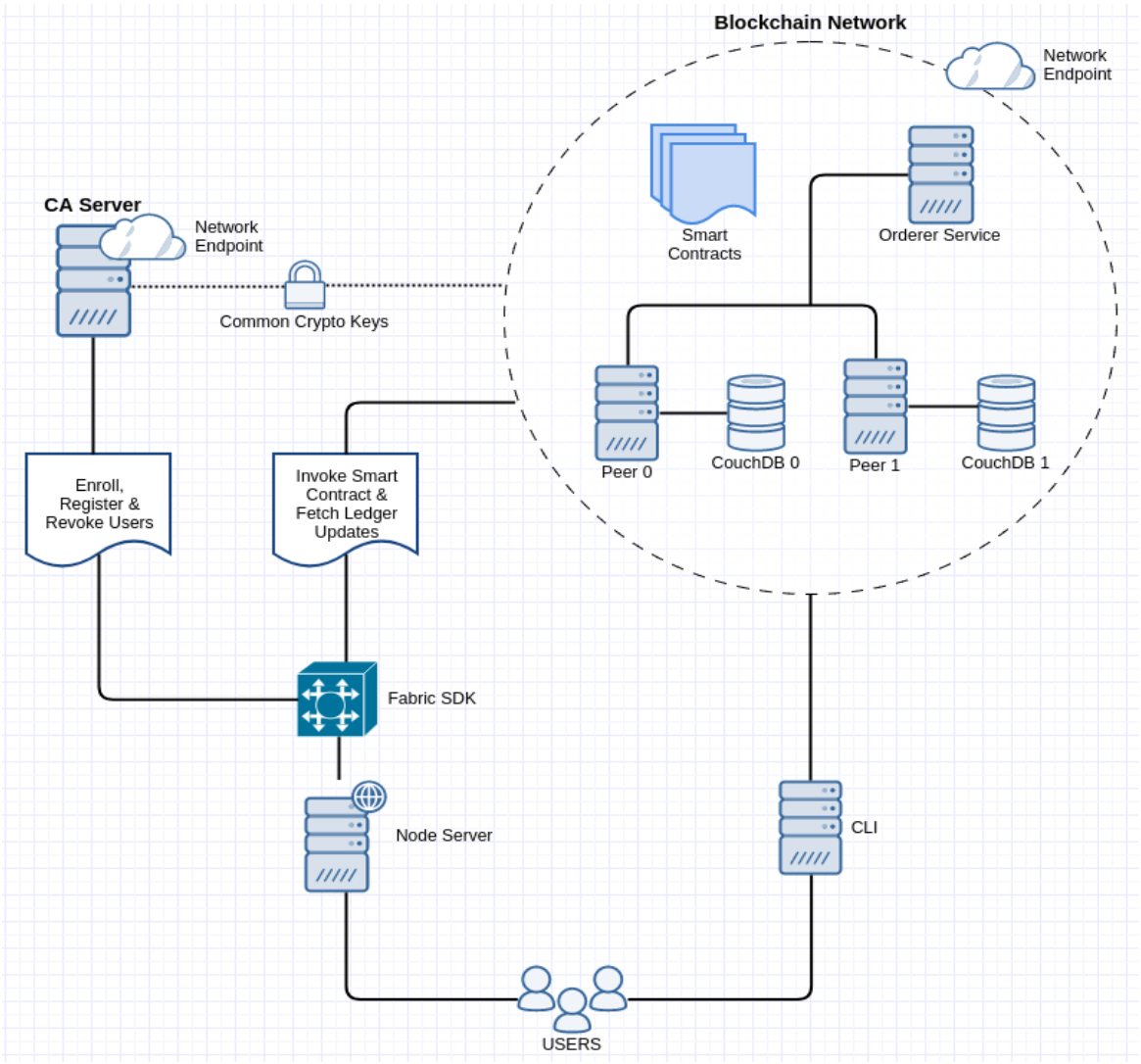

Рисунок 4.1 - Приклад простої мережі

Як видно з рисунку 4.1, кожна мережа складається з базових елементів, серед яких є peers, order service, ca server. Для збереження даних у мережі блокчейну використовується ChouchDB. Для підключення до мережі можна використовувати Fabric sdk, який надає api для взаємодії з мережею блокчейну.

Наступним неменш важливим елементом мережі є Ordering service. Замовники формують послугу замовлення, тобто комунікаційну транзакцію, яка забезпечує гарантії доставки даних до кожного із створених вузлів мережі. Послуга замовлення може бути реалізована різними способами: від централізованого сервісу (наприклад, у розробці та тестуванні) до розподілених протоколів, які орієнтовані на різні види мережевих і вузлових помилок[17].

Служба замовлень надає спільний канал зв'язку клієнтам і peers, пропонуючи послуги розсилки для повідомлень, що містять транзакції. Клієнти підключаються до каналу і можуть транслювати повідомлення на каналі, які потім доставляються до всіх однорангових каналів. Канал підтримує доставку всіх повідомлень, із повним підтвердженням доставки та надійності. Іншими словами, канал виводить одне й те саме повідомлення до всіх підключених однорангових вузлів і доставляє їх всім одноранговим вузлам в одному логічному порядку. Ця гарантія зв'язку також називається трансляцією повного порядку, atomic broadcast або консенсусом у контексті розподілених систем. Доставлені повідомлення є кандидатами для включення в стан блокчейну.

Наступним важливим елементом даної мережі є CA Server. Hyperledger Fabric CA є центром сертифікації (CA) для Hyperledger Fabric. Вона надає такі функції, як: реєстрація ідентифікаторів або підключення до LDAP, оскільки реєстр користувачів видачі сертифікатів про реєстрацію (ECerts) поновлення сертифіката і анулювання Hyperledger Fabric CA складається як з сервера, так і з клієнтських компонентів[16]. Таким чином дана система надає користувачам можливість авторизуватися в мережі блокчейну, щоб отримати відповідні переваги для подальших дій у даній мережі.

Оглянувши базові елементи даної мережі на цьому етапі, опишемо вище наведену схему базової мережі. Як видно з рисунку 3.1, мережа складається з 2 peers, кожен з яких містить свою базу даних, котра у свою чергу зберігає в собі актуальну копію журналу. Також із рисунка можна відмітити, що дана мережа містить тільки один сервіс, який слідкує за актуальністю журналу та одного серверу сертифікатів. Для взаємодії з мережею користувачі можуть використовувати набір бібліотек або термінал, на пряму підключившись до блокчейну.

4.2 Документо-орієнтована система управління базами даних Chouch DB

Для збереження транзакцій і даних із контрактів потрібна база даних, тому Hyperledger Fabric підтримує два типи однорангових баз даних. LevelDB - це база даних котра використовується фреймворком за замовчуванням, вона вбудована в

вузол, і зберігає дані ланцюгових кодів, як прості пари ключ-значення, підтримує тільки ключові запити[17]. Користувач LevelDB має можливість перевизначити метод сортування даних, вказавши власну функцію порівняння, що надає можливість гнучкого сортування даних.

CouchDB є необов'язковою базою даних альтернативного стану, яка підтримує розширені запити, коли значення ланцюгових кодів моделюються як JSON. JSON-запити є більш гнучкими та ефективними щодо великих індексованих сховищ даних, коли ви хочете запитувати фактичний вміст значення даних, а не ключі. CouchDB є сховищем документів JSON, а не чистим сховищем ключів, що дозволяє індексувати вміст документів у базі даних.

Щоб використовувати переваги CouchDB, а саме JSON-запити на основі вмісту, ваші дані повинні бути змодельовані у форматі JSON. Ви повинні вирішити, чи слід використовувати LevelDB або CouchDB, перш ніж налаштувати мережу. Перемикання вузла з використанням LevelDB на CouchDB не підтримується через проблеми із сумісністю даних. Усі вузли мережі повинні використовувати один і той же тип бази даних. Якщо у вас є суміш JSON і бінарних даних, ви все ще можете використовувати CouchDB, однак двійкові значення можна запитувати лише на основі ключів, діапазону ключів і складених ключових запитів. Таким чином використання даного типу бази даних при проектуванні надає розробнику додаткові можливості, що в подальшому можуть полегшити рішення проблем, що можуть з’явитися.

4.3 Проектування архітектури мережі

При розробці системи на основі блокчейну одним із основних етапів є проектування мережі, з якою в подальшому буде взаємодіяти розроблювана система. Завдяки якій користувач системи, у подальшому матиме можливість зберігати свої наукові роботи з гарантією їхньої цілісності. Для реалізації цього етапу розробки, створено архітектурну схему мережі, яка зображена на рисунку 4.2.

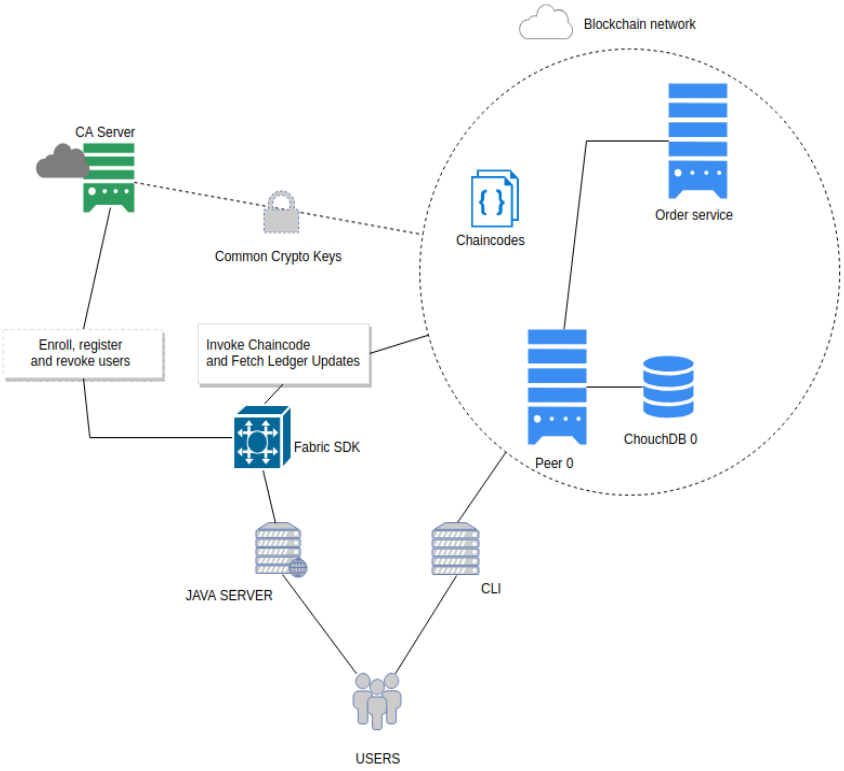

Рисунок 4.2 - Схема мережі

Як видно з рисунку 4.2, розроблювана система буде містити тільки один peer, який для зберігання даних журналу використовуватиме chouchDB, котрий у свою чергу буде взаємодіяти з одним order service. Таким чином при потраплянні даних на сервер замовлень, у подальшому потрібно буде оновити тільки один peer. Для автентифікація користувачів у даній мережі було розміщено один сервер сертифікатів, за допомогою якого, використовуючи java sdk, будуть відбуватися всі подальші дії в блокчейні.

Висновки до розділу

У даному розділі було проведено розробку архітектури мережі блокчейн для майбутньої системи та розглянуто базові компоненти мережі Hyperledhger Fabric, послідовність взаємодії кожного з компонентів даної мережі. У результаті було спроектовано мережу, котра буде використовуватися системою.

5. ТЕСТУВАННЯ РОБОТИ СИСТЕМИ

При розробці програмного продукту для запуску та тестування застосунку використовується docker, котрий у свою чергу являється програмою для автоматизації розгортання і управління застосунками в середовищах із підтримкою контейнеризації. Тому першим етапом для тестування роботи програми потрібно встановити в систему docker та docker-compose. Після встановлення яких тестувальник повинен виконати в консолі команду “docker-compose up –build”, за допомогою якої він може запустити програмний продукт.

Після запуску програмного продукту користувач системи потрапляє на головну сторінку сайту, котра зображена на рисунку 5.1, на якій користувач системи може спостерігати основну інформацію, яка описує базові можливості цього програмного продукту, також на сторінці міститься головне меню, котре надає користувачу можливість здійснювати навігацію по сайту.

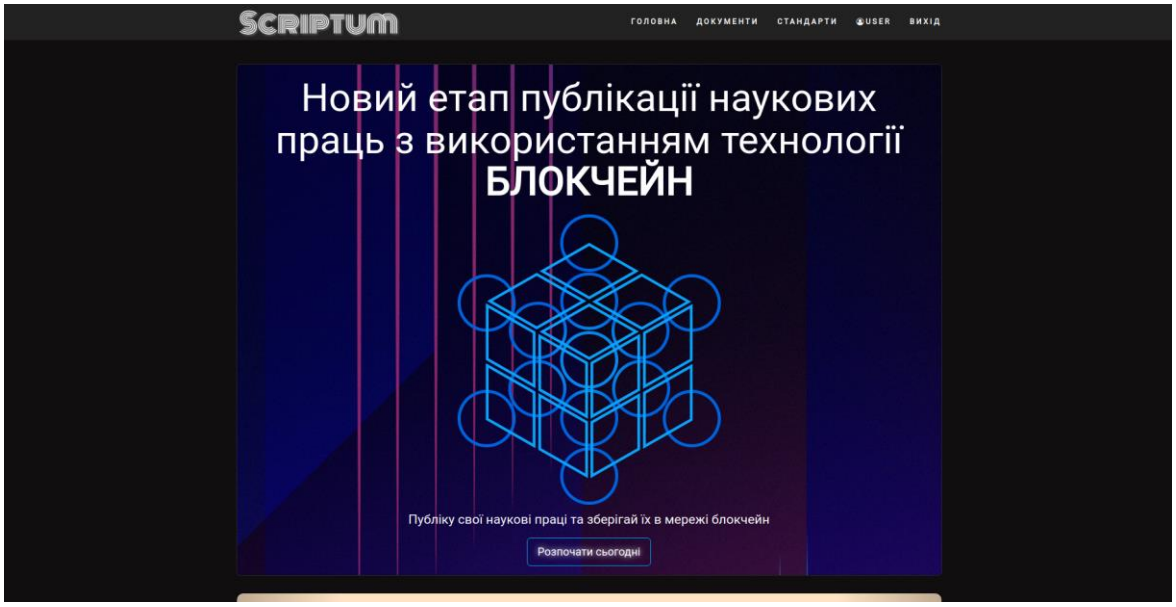

Рисунок 5.1 – Головна сторінка сайту

Після того як користувач системи авторизується в ній, він потрапляє в панель користувача, котра зображена на рисунку 5.2, яка надає можливість йому додавати документи в веб-застосунок, також редагувати та видаляти їх.

Авторизований користувач окрім роботи з документами, також може здійснювати налаштування свого аканту перейшовши на сторінку “Налаштування”.

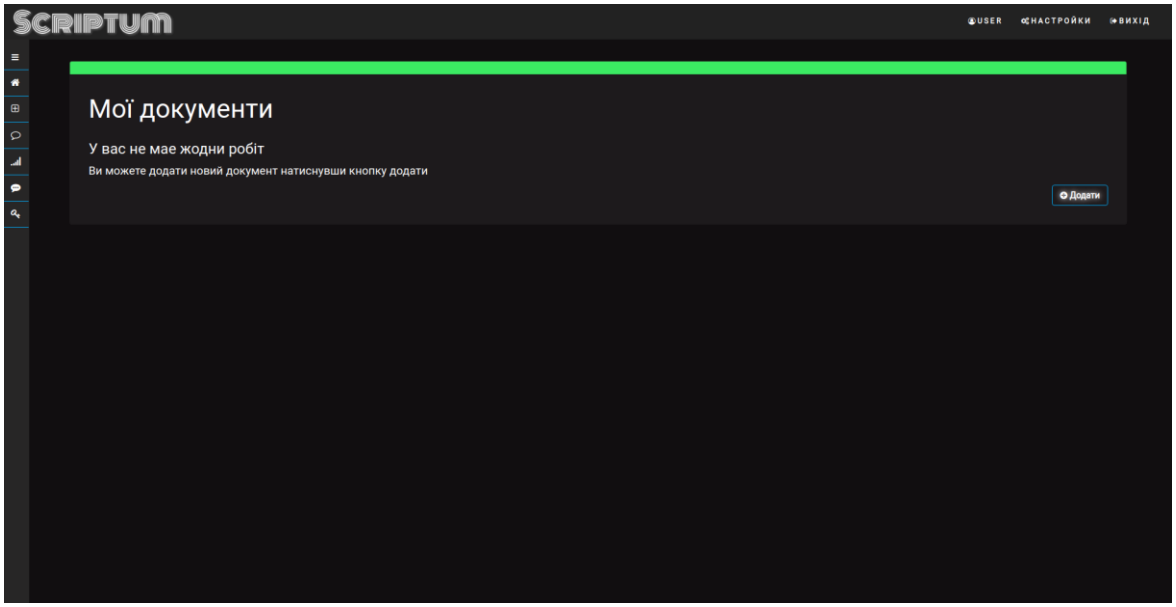

Рисунок 5.2 – Персональна сторінка користувача

Після того як користувач натисне кнопку «Додати», він перейде на сторінку додавання нового документу до системи. Як видно із рисунку 5.3, користувач системи має можливість вказувати назву документа, короткий опис самого документа, можливість вибору типу документа та предметної області самого документа. Після заповнення всіх полів і натиснення кнопки «Додати», дані будуть відправлені на сервер, де їх буде перевірено та додано до бази даних.

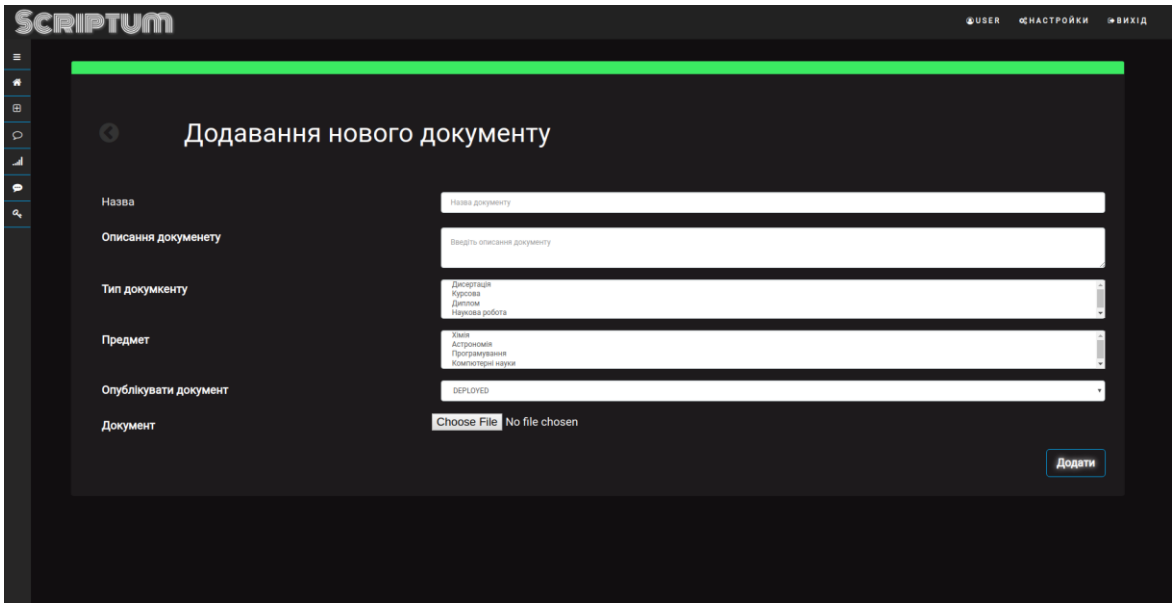

Рисунок 5.3 – Додавання нового документу

Кінцевим етапом додавання нового документу є поява його на головній сторінці в профілі користувача системи, як це зображено на рисунку 5.4. На котрій

вже для доданих документів користувач системи має можливість редагувати та видаляти їх.

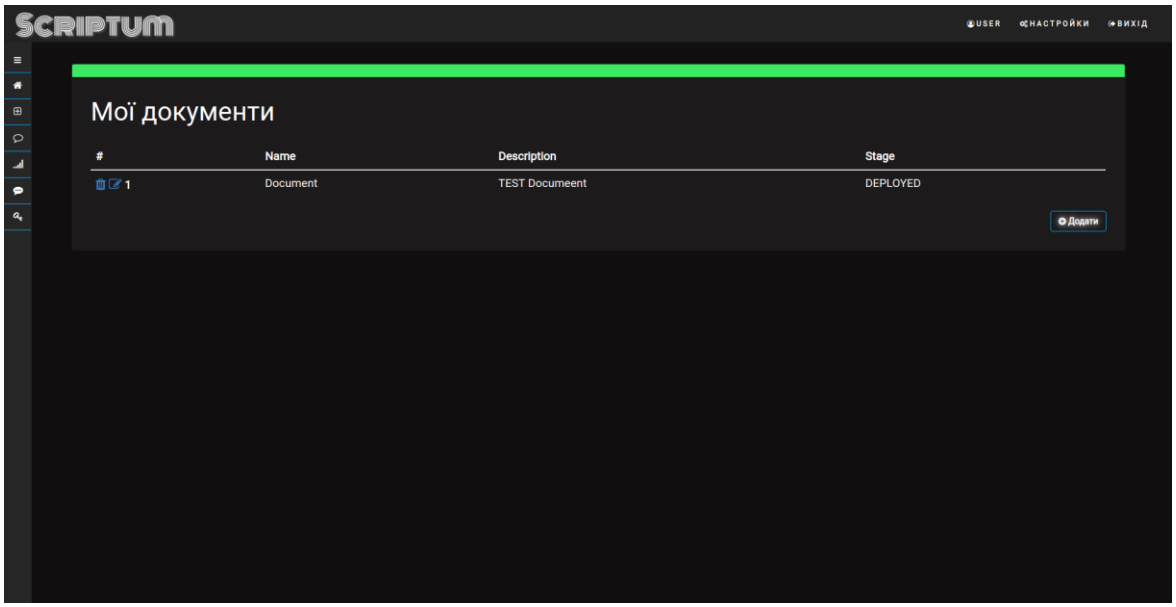

Рисунок 5.4 – Додавання нового документу

Наступним і одним із важливих елементів після додавання документу є його перегляд іншими користувачами системи. Для цього розробником було реалізовано сторінку із документами, котра зображена на рисунку 5.5.

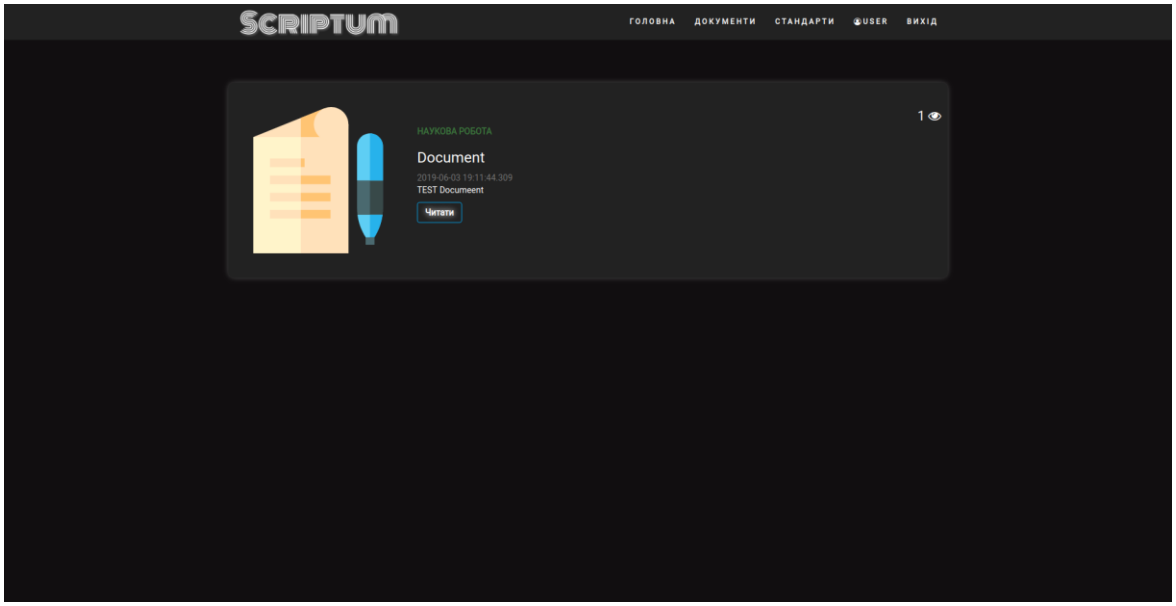

Рисунок 5.5 – Документи додані користувачами

Як видно з рисунку 5.5, для того щоб переглянути документ користувачу системи потрібно натиснути кнопку «Читати», після натиснення котрої користувач потрапляє на сторінку зображену на рисунку 5.6.

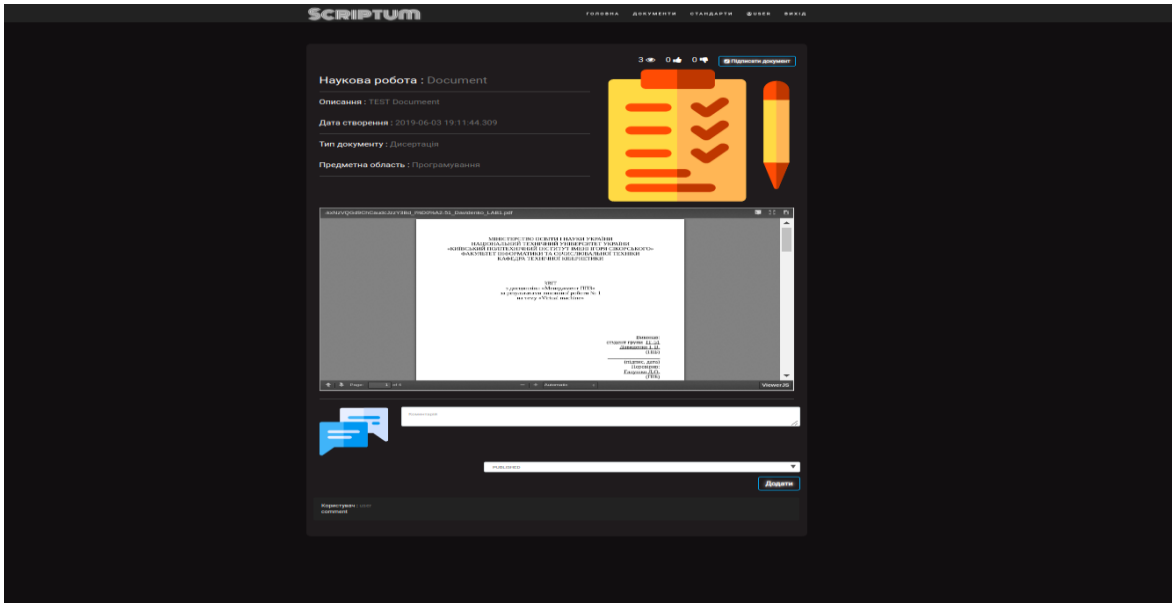

Рисунок 5.6 – Перегляд документу користувача

Я видно з рисунку 5.6, на цій сторінці авторизований користувач системи має можливість переглядати короткий опис документа та читати його. Після закінчення перегляду користувачем системи, він має можливість висловити свою думку в коментарях до документу та також поставити лайки та дизлайки, таким чином даний зворотній зв'язок надасть можливість авторам документа завжди бачити свої помилки та мати можливість їх вчасного виправлення.

Таким чином у процесі тестування даного модулю системи було виявлено декілька помилок та їх виправлено, також для більшої ясності, які із модулів вже було протестовано, розробник системи розробив матриця трасуємості.

5.1 Матриця трасуємості (traceability matrix)

У процесі тестування одним із головних етапів є розподілення задач, є контроль цілісності програмного продукту, що розробляється, виявлення розбіжностей між ТЗ і реалізованим функціоналом. Матриця трасуємості являє собою двовимірну таблицю, котра містить відповідність функціональних вимог (functional requirements) продукту і підготовлених тестових сценаріїв (test cases). У заголовках колонок таблиці розташовані вимоги, а в заголовках рядків - тестові сценарії. На перетині - відмітка, що означає, що вимога поточної колонки покрито

тестовим сценарієм поточного рядка. Матриця зазвичай зберігається у вигляді електронної таблиці. Дана матриця зображена в таблиці 6.1.

Таблиці 5.1 - Traceability matrix

	Тест				
		Запис зберігаєтьс я в базі даних	Форма виводитьс я на тій же сторінки	Виводиться помилка при відсутності обов'язкових полів	Виводиться повідомлення про успішне додавання елементу
	Заповнення форми реєстраціїї		V	V	
	Обробка форми реєстрації	V			
	Повернення результатів реєстрації				V
	Заповнення форми документу		V	V	
	Додавання документу	V			V
	Заповнення форми коментарю		V	V	
	Додавання коментарю	V			

Продовження таблиці 5.1

	Додавання підпису до документу	V			V

Таким чином матриця трасуємості може служити одночасно в якості матриці покриття. Наявність такої матриці дозволяє об'єктивно оцінити, яка частина продукту покрита тестами, а яка ні. Це необхідна умова, щоб оцінити, який обсяг роботи було виконано розробниками веб-застосунку, і що ще залишилося зробити - і по частині створення, і по частині виконання тестів.

Висновки до розділу

У розділі було проведено тестування створеної системи, а саме спроектованих та реалізованих модулів, яке здійснювалось в автоматизованому та в ручному режимах. Для кращого виявлення помилок при роботі програми було застосовано логування виникаючих помилок.

У результаті було визначено, що перший модуль розроблюваної система повністю протестований та готовий до експлуатації.

ВИСНОВКИ

У результаті виконання дипломного проекту на тему «Програмний засіб зберігання наукових праць за технологією блокчейн» було розроблено веб-застосунок та досягнуто мети дипломного проекту.

Проведено дослідження рішень для побудови застосунків із використанням розподіленої архітектури та розглянуто платформи для побудови мережі блокчейну.

За результатами досліджень проведених у процесі виконання дипломного проекту було обрано рішення для проектування застосунку з використанням розподіленої архітектури та обрано мережу блокчейну, котра буде забезпечувати зберігання наукових робіт, також для реалізації мережі обрано Hyperledger Fabric, котра надає можливість розробити власний приватний блокчейн.

У процесі розробки системи було пройдено повний цикл розробки застосунку, також розроблено веб-застосунок, котрий надає можливість зберігати наукові роботи в мережі блокчейн.

Була досягнута поставлена мета у вигляді розробленого веб-застосунку та спроектованої, підключеної мережі блокчейну.

Розроблена система відповідає всім заявленим вимогам, також всі поставлені задачі було виконано, протестовано веб-застосунок. Система реалізована в повному обсязі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Architecture Origins [Електронний ресурс] – режим доступу до ресурсу: https://hyperledger-fabric.readthedocs.io/en/release-1.4/arch-deep-dive.html – Назва з екрану
2. Hibernate (библиотека) [Електронний ресурс] – режим доступу: https://ru.wikipedia.org/wiki/Hibernate – Назва з екрану
3. База даних [Електронний ресурс] – режим доступу: https://uk.wikipedia.org/wiki/База_даних – Назва з екрану
4. Что такое база данных [Електронний ресурс] – режим доступу: http://www.codenet.ru/progr/vbasic/vb_db/1.php – Назва з екрану
5. Триярусна архітектура [Електронний ресурс] – режим доступу: https://uk.wikipedia.org/wiki/Триярусна_архітектура – Назва з екрану
6. SQLite, MySQL и PostgreSQL [Електронний ресурс] – режим доступу: https://tproger.ru/translations/sqlite-mysql-postgresql-comparison/ – Назва з екрану
7. Архитектура веб-приложений. Стек Spring MVC + AngularJsPostgreSQL [Електронний ресурс] – режим доступу: https://habrahabr.ru/company/piter/blog/269217/ – Назва з екрану
8. Как писать на Spring в 2017 [Електронний ресурс] – режим доступу: https://habrahabr.ru/post/333756/ – Назва з екрану
9. Руководство по Hibernate. Архитектура [Електронний ресурс] – режим доступу: http://proselyte.net/tutorials/hibernate-tutorial/architecture/ – Назва з екрану
10. Руководство по Hibernate. Язык запросов Hibernate (HQL) [Електронний ресурс] – режим доступу: http://proselyte.net/tutorials/hibernate-tutorial/hibernate-query-language/ – Назва з екрану
11. The Base16, Base32, and Base64 Data Encodings [Електронний ресурс] - режим доступу : https://tools.ietf.org/html/rfc4648#page-5 - Назва екрану

12. Кодування із стисненням інформації [Електронний ресурс] - режим доступу : https://web.posibnyky.vntu.edu.ua/firen/6bilynskyj_elektronni_systemy/56.htm - Назва екрану
13. Структура бази даних [Електронний ресурс] - режим доступу : https://studopedia.com.ua/1_56146_struktura-bazi-danih.html - Назва екрану
14. БЛОК-СХЕМА [Електронний ресурс] - режим доступу : https://studbooks.net/2114673/informatika/blok_shema - Назва екрану
15. Вішал Лайка «Learn Java for Web Development» / Вішал Лайка – К. : «Apress», 2014. – 461 с.
16. Fabric CA User’s Guide [Електронний ресурс] - режим доступу : https://hyperledger-fabric-ca.readthedocs.io/en/latest/users-guide.html - Назва екрану
17. CouchDB as the State Database [Електронний ресурс] - режим доступу : https://hyperledger-fabric.readthedocs.io/en/latest/couchdb_as_state_database.html - Назва екрану
18. Задача византийских генералов [Електронний ресурс] - режим доступу : https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%B2%D1%96%D0%B7%D0%B0%D0%BD%D1%82%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B8%D1%85_%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D0%BB%D1%96%D0%B2 - Назва екрану
19. Что такое алгоритмы консенсуса, Proof-of-Work (PoW), Proof-of-Stake (PoS) и другие [Електронний ресурс] - режим доступу : http://bitstat.top/blog.php?id_n=2152 - Назва екрану
20. Sqlite vs mysql vs postgresql порівняння систем управління базами даних - вибір бази даних [Електронний ресурс] - режим доступу : http://jak.magey.com.ua/articles/sqlite-vs-mysql-vs-postgresql-porivnjannja-sistem.html - Назва екрану